AD-753 120

FEABL (FINITE ELEMENT ANALYSIS BASIC LIBRARY) USER'S GUIDE

Oscar Orringer, et al

Massachusetts Institute of Technology

Prepared for:

Air Force Office of Scientific Research

August 1972

DISTRIBUTED BY:

NTIS
National Technical Information Service
U. S. DEPARTMENT OF COMMERCE
5285 Port Royal Road, Springfield Va. 22151

AFOSR TR 72-2228 ASRL TR 162-3

FEABL (FINITE ELEMENT ANALYSIS BASIC LIBRARY) USER'S GUIDE

Oscar Orringer
Susan E. French

AEROELASTIC AND STRUCTURES RESEARCH LABORATORY
DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
CAMBRIDGE, MASSACHUSETTS 02139

AUGUST 1972

Prepared for

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
UNITED STATES AIR FORCE

Under
CONTRACT NO F44620 70 C 0020

Approved for public release,
distribution unlimited.

NOTICES

Qualified requestors may obtain additional copies from the Defense Documentation Center, all others should apply to the National Technical Information Service.

Reproduction, translation, publication, use, and disposal in whole or in part by or for the United States Government is permitted.

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
AEROELASTIC & STRUCTURES RESEARCH LABORATORY
CAMBRIDGE, MASSACHUSETTS 02139

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP

3. REPORT TITLE
FEABL (FINITE ELEMENT ANALYSIS BASIC LIBRARY) USER'S GUIDE

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Scientific Interim

5. AUTHOR(S) *(First name, middle initial, last name)*
OSCAR ORRINGER SUSAN E FRENCH

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
Aug 1972	105	5

8a. CONTRACT OR GRANT NO. F44620-70-C-0020
b. PROJECT NO. 9782-02
c. 61102F
d. 681307

9a. ORIGINATOR'S REPORT NUMBER(S)
ASRL-TR-162-3

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
AFOSR-TR-72-2228

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited

11. SUPPLEMENTARY NOTES
TECH, OTHER

12. SPONSORING MILITARY ACTIVITY
AF Office of Scientific Research (NAM)
1400 Wilson Blvd.
Arlington, Va. 22209

13. ABSTRACT
This guide contains complete instructions for the use of FEABL, a basic software system developed for finite element analysis at the MIT Aeroelastic and Structures Research Laboratory. FEABL has been designed primarily for (but is not limited to) the specialized type of continuum analysis problem encountered in the materials laboratory. FEABL has also been used successfully as an educational tool in the finite element analysis course given by the MIT Department of Aeronautics and Astronautics. The software is modular, and is written in machine-independent FØRTRAN IV. A complete program listing is contained in Appendix C. Element subroutines which can be used either independently or in conjunction with FEABL will be presented in future publications.

IA

UNCLASSIFIED

UNCLASSIFIED

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
FINITE ELEMENT ANALYSIS						
STRUCTURAL ANALYSIS						
NUMERICAL METHODS						
FINITE ELEMENT SOFTWARE						
MATRIX METHODS						

I B

UNCLASSIFIED

Security Classification

AFOSR TR
ASRL TR 162-3

FEABL
(FINITE ELEMENT ANALYSIS BASIC LIBRARY)
USER'S GUIDE

Oscar Orringer
Susan E. French

August 1972

Prepared for

Aeromechanics Division
Air Force Office of Scientific Research
Arlington, Virginia 22209
Under
CONTRACT NO. F44620-70-C-0020

Approved for public release; distribution unlimited.

Aeroelastic and Structures Research Laboratory
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

FOREWORD

The developments documented in this report were carried out at the Aeroelastic and Structures Research Laboratory, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, Massachusetts under Contract No. F44620-70-C-0020 from the Office of Scientific Research, U.S. Air Force. The software system originated in connection with Dr. Orringer's need to use finite-element analysis for problems in the compressive behavior of fiber composites that were being investigated under the Contract. Dr. Jacob Pomerantz of the Aeromechanics Division, AFOSR served as technical monitor.

The authors wish to express their appreciation to Prof. Pin Tong of the Department of Aeronautics and Astronautics for the numerous helpful suggestions he made regarding the design of the FEABL software system. The authors are also indebted to all of the students in the Departments 1971-72 finite element analysis course who, under Prof. Tong's guidance, acted as guinea pigs while the software was still being tested. For his review of this report and for attendant constructive suggestions, the authors are indebted to Professor Emmett A. Witmer of the MIT-ASRL.

The computations and testing of the program were carried out at the MIT Information Processing Center.

ABSTRACT

This guide contains complete instructions for the use of FEABL, a basic software system developed for finite element analysis at the MIT Aeroelastic and Structures Research Laboratory. FEABL has been designed primarily for (but is not limited to) the specialized type of continuum analysis problem encountered in the materials laboratory. FEABL has also been used successfully as an educational tool in the finite element analysis course given by the MIT Department of Aeronautics and Astronautics. The software is modular, and is written in machine-independent FØRTRAN IV. A complete program listing is contained in Appendix C. Element subroutines which can be used either independently or in conjunction with FEABL will be presented in future publications.

CONTENTS

CONTENTS (Continued)

SECTION 1

INTRODUCTION

1.1 Purpose and Scope

Finite element analysis programs developed in the past fall into two general categories: the "one-shot" program optimized for solution of a particular problem, or the "systems" program designed for the non-optimum solution of any problem. The structural engineer who has a new analysis to perform must spend a great amount of time either writing his own "one-shot" program or learning how to use one of the general "systems". FEABI attempts to relieve the analyst of these burdens by providing a basic library which handles the numerical operations common to all finite element analysis, and which can be understood and used after a few hours of study. FEABL's current capabilities are limited to static analysis. Other options will be added in the future, as the need arises, and according to their usefulness. This guide is written for the analyst who has some acquaintance with the theory of finite element analysis and with IBM FØRTRAN IV.

THE FEABL software system has been designed primarily for "materials laboratory" stress analysis. The analysis problems encountered in this application area are characterized by geometry which, although not simple, is mathematically describable, and by stress solution accuracy requirements which can usually be met with models of 1,000 or fewer degrees of freedom. Stress concentration analysis for two-material systems and for composite-to-metal step-lap joints are two examples of the materials laboratory type of problem. Another feature of many of these problems is a requirement for nonstandard solution methods. For instance, the boundary conditions may include an undetermined prescribed displacement, the value of which depends upon satisfaction of an auxiliary condition on a boundary stress integral; a specific example is given in Ref. 1. The main features of FEABL, all-in-core computation and control of the solution process by the user, are well adapted to the needs of such problems.

Experience in the MIT Department of Aeronautics and Astronautics has also shown that FEABL is a valuable educational tool. A budget of $50 to $70 per man on an IBM 370/155 machine allows each student to solve one or two problems, each having 200 to 300 degrees of freedom. In the approach taken at MIT, the student is required to program his structure geometry, element interconnections, and subroutines for generating stiffnesses and stresses. He must also achieve a general understanding of the principles upon which the FEABL subroutines are based. Thus, each student is able to gain some practical experience, as well as theoretical knowledge in one semester of a finite element analysis course. This approach is felt to be superior to the traditional method of having the student learn only how to code the input data for a particular, complete "systems" program.

The flexibility and economy of the FEABL software system has been amply demonstrated by various applications in which real problems were solved during the test/exercise development phase. The following are examples abstracted from these applications:

Problem Description	Total DOF	CPU Time (Min)	Total Cost ($)
Displacement analysis of two large rigid frame structures (6 load cases each)	216 426	0.42 2.49	5.37 13.17
Continuum stress concentration analysis with boundary condition determined by auxiliary stress integral (3 complete solutions required for each case)	338 442	1.50 2.70	4.50 12.00
Stress analysis of a large frame in an oil tanker, using rectangle and triangle continuum elements and flange elements (6 loading cases)	1,266	3.0 (Approx.)	14.00
Pressure distribution analysis for viscous flow around a corpuscle in a blood vessel	160	No data available	
Stress analysis, including solution for stress intensity factor of a sharp crack, using hybrid element at the crack	160	0.13	2.54

The above examples were run on the IBM 370/155 at the MIT Information Processing Center. The costs presented are for production runs, exclusive of debugging.

1.2 General System Concepts

Development of the FEABL software has been based on three general concepts. First, FØRTRAN IV was selected as the program language, and all FØRTRAN syntax rules were followed rigorously. This makes FEABL as machine-independent as possible, to allow for use at installations having other than IBM equipment. The only restrictions are that the FØRTRAN compiler software must permit logical IF statements, variable dimensioning, six-character variable names, and labelled CØMMØN declarations. Second, the software is modular in character. The finite element analysis process has been decomposed into a series of distinct steps, and each step which is independent of structure geometry has been programmed as a separate subroutine. FEABL is thus analogous, in a sense, to the IBM Scientific Subroutine Package.

The third concept deals with data storage techniques. In order to minimize wasted storage space, a vector approach has been adopted. All of the input data for a problem, certain "housekeeping" data, and internally generated problem data such as the master stiffness matrix are stored in a single vector, referred to below as the /DATA/ vector. In addition, control parameters which locate entries in the /DATA/ vector or manage input-output operations are organized in four CØMMØN groups.

1.3 General System Organization

The analyst must provide one or more programs which interface with the FEABL software:

1. A MAIN program which inputs the required problem data and controls the execution of all sub ›rograms.
2. One or more subroutines which generate an element stiffness matrix (and, if required, equivalent nodal forces for thermal stress, gravity, etc.) for each type of element.*

*Subroutines from ASRL EGL (Element Generator Library) may be used.

3. One or more subroutines which generate element stresses from the element's nodal displacements.*

Data may be input by reading cards, disk or tape files, by automatic generation techniques, or by a combination of the two methods. The details of a particular problem will determine which approach is more efficient. For very simple types of problems, in which the structure is divided into only a few different element types and shapes, stiffness matrix and stress generation may be done within the MAIN program. However, if separate subroutines are required for these tasks, pre-existing ones may be interfaced with FEABL, since the FEABL software package has been written in a manner such that pre-existing FØRTRAN subroutines may be adapted to run with FEABL with little or no re-programming.

The interface between user-written programs and FEABL consists of two parts: data location and process control. Figure 1 illustrates the general features of the data location interface. The labelled CØMMØN areas mentioned in the previous section must appear in the user's MAIN program. This is accomplished simply by including identical sets of DIMENSIØN, CØMMØN and EQUIVALENCE declarations in MAIN and in the FEABL software. There is no requirement for the element generator subroutines to communicate with FEABL in this manner; these subroutines need only be interfaced with user's MAIN by an identical DIMENSIØN declaration for the element stiffness matrix and "force/displacement" vector. The latter is a dual-purpose vector in which element equivalent nodal forces are generated, and which serves later as a storage area for the solution displacement vector of the element. The interface between the generator routines and FEABL is achieved automatically by variable DIMENSIØN declarations contained in the FEABL software. The data location interface is discussed in detail in Section 2.

*Subroutines from ASRL EGL (Element Generator Library) may be used.

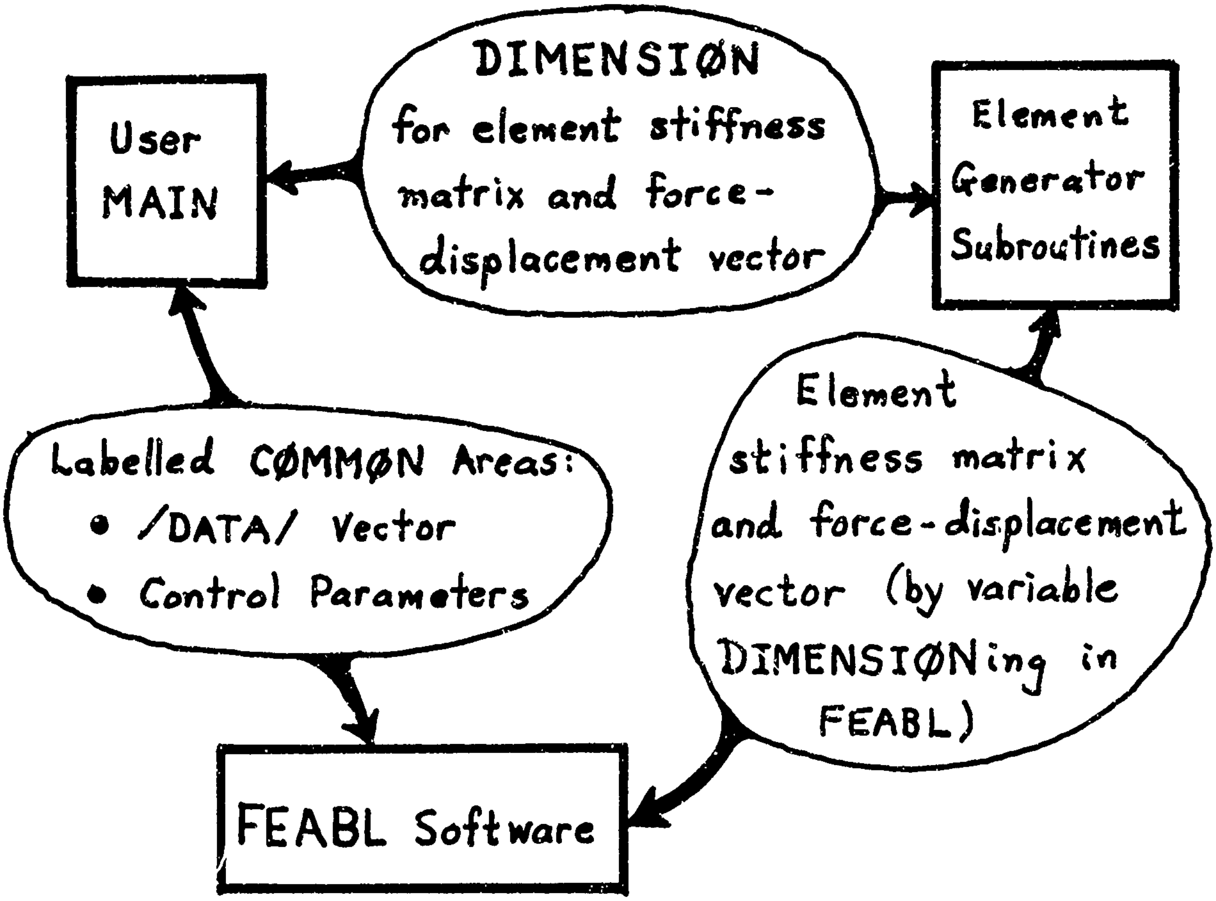


FIGURE 1

The process control interface is simply a recognition that input of problem data and operations on the data must be performed in a certain sequence. This requires a definite series of programming steps in user's MAIN program. Section 3 discusses the process control interface in detail.

SECTION 2

THE FEABL DATA STORAGE SYSTEM

2.1 Accessory Data

The accessory data is divided into four control parameter CØMMØN groups. The CØMMØN declaration statements listed below for the four control parameter groups must appear in all user-written MAIN programs.

2.1.1 Input-Output Control Parameters

CØMMØN /IØ/ KR, KW, KP, KT1, KT2, KT3

The six parameters in the /IØ/ group are hardware device codes, defined as follows:

KR - card reader

KW - printer

KP - card punch

KT1, KT2, KT3 are extra device codes which the user may define optionally as he pleases. For example, KT1 might refer to a system direct access disk, KT2 to an external tape unit, etc. Only the printer device code KW is used directly by the FEABL software presented in this guide. The input-output control parameters are used in FØRTRAN READ and WRITE statements, e.g.:

```
    READ (KR,1) NET, NDT
  1 FØRMAT (2I6)
    WRITE (KW,2) NET, NDT
  2 FØRMAT (16HOTØTAL ELEMENTS=, I6, 10X, 10HTØTAL DØF=, I6)
```

The correct values of KW and any other device codes employed by the user must be established at the beginning of the MAIN program.

2.1.2 Problem Size Parameters

CØMMØN /SIZE/ NET, NDT

These parameters define the problem size in terms of the total number of elements (NET) and the total number of degrees of freedom (NDT) in the whole structure. NDT includes both constrained and unconstrained degrees. In standard solutions these parameters are established at the beginning of the MAIN program and remain fixed.

2.1.3 Address Index Parameters

```
COMMØN /BEGIN/ ICØN, IKØUNT, ILNZ, IMASTR, IQ, IK
COMMØN /END/ LCØN, LKØUNT, LLNZ, LMASTR, LQ, LK
```

These parameters control the begin and end locations of data sub-blocks in the /DATA/ vector. The user establishes correct values for these parameters by calling FEABL subroutine SETUP after he has established the /SIZE/ block. Use of the address index parameters to locate entries in the /DATA/ vector is discussed in Subsection 2.2.

2.2 The /DATA/ Vector

```
DIMENSIØN REAL(xxxx), INTGR(xxxx)
CØMMØN /DATA/ REAL
EQUIVALENCE (REAL (1), INTGR(1))
```

The above declarations serve to define the /DATA/ vector as a one-dimensional array occupying a CØMMØN area labelled /DATA/ and having two reference names: REAL for floating point entries and INTGR for integer entries. These declarations must appear in the user-written MAIN program. The user chooses the dimension integer "xxxx" to suit his particular needs. The DIMENSIØN declaration with user's value for the length of the /DATA/ vector must be inserted in each FEABL subprogram, immediately following the subroutine name declaration, e.g.:

```
SUBRØUTINE ASEMBL (LNUM, NDE, ELK, ELQ)
DIMENSIØN REAL(1000), INTGR(1000)
```

Some compilers restrict the length of any vector to 32,768 words ($8,000_{HEX}$). If this restriction is encountered, and the user desires a longer /DATA/ vector, say 50,000 words, the following form of the declaration statements can be employed:

```
DIMENSIØN REAL(25000), INTGR(25000), DUMMY(25000)
CØMMØN /DATA/ REAL, DUMMY
EQUIVALENCE (REAL(1), INTGR(1))
```

The CØMMØN /DATA/ declaration must then be changed to the above form in each FEABL subprogram.

The dual nature of the /DATA/ vector must be recognized clearly. For example, the 100th entry in the vector may be treated as either a floating point or an integer quantity in arithmetic or logical instructions by operating, respectively, with REAL(100) or INTGR(100). This property will lead to compilation errors if the user attempts to save storage space by declaring INTGR to be an array of 2-BYTE words (half-words), while REAL is left as an array of normal 4-BYTE words (full words), or if the user attempts to run in double precision mode for floating point arithmetic.

The /DATA/ vector is organized into six blocks, the limits of which are determined by the values stored in the address index parameters (Subsection 2.1.3). In normal usage, the first four blocks contain integer data and the last two contain floating point data. Storage conventions are detailed in the following sections, in the order in which the blocks appear in the /DATA/ vector.

2.2.1 Constraint Vector

This is an integer block which contains the global number of each degree of freedom at which a displacement is to be prescribed. Entries in the constraint vector are referred to by:

INTGR(I) where ICØN≤I≤LCØN

The following example illustrates user action involving the constraint vector

```
•
•
•
INTGR(ICØN)=2
INTGR(ICØN+1)=25
INTGR(LCØN)=6
•
•
•
```

By means of the above instructions the user has specified that global displacements 2, 25, and 6 will be prescribed. The prescribed displacements need not be listed in any particular order. Any excess space in the constraint vector is filled with zeros by FEABL.

2.2.2 Address Count Vector

This is an integer block which contains information relating to the absolute address in the /DATA/ vector of each diagonal entry of the master (global) stiffness matrix. Entries in the address count vector are referred to by:

INTGR(I) where IKØUNT$\leq$I$\leq$LKØUNT

To obtain the relevant data for the Nth row, the correct subscript is:

J=IKØUNT+N-1
INTGR(J) contains address information for row N

Normally, the user has no direct communication with the address count vector; its contents are generated internally by FEABL. The absolute address of each diagonal entry is output for use in debugging. After the output, the data in the address count vector are modified as follows:

Contents = (Absolute Address of K_{ii}) - i

Subsequently, any stored* entry K_{ij} can be obtained by referring to:

REAL(KADR)

where the address is given by:

KADR = INTGR(IKØUNT+I-1)+J

2.2.3 Leading Non-Zero Entry Vector

The LNZ vector is an integer block containing the number j of the column in which appears the leading non-zero entry K_{ij}

*Many entries of the master stiffness matrix are not stored in the /DATA/ vector. See Subsection 2.2.6 for details.

of the master stiffness matrix for each row i. Entries in the LNZ vector are referred to by:

INTGR(I) where $ILNZ \leq I \leq LLNZ$

INTGR(ILNZ+N-1) refers to row N.

Normally, the user has no direct communication with the LNZ vector; its contents are generated internally by FEABL, and are output for use in debugging and evaluating displacement numbering strategies.

2.2.4 Master Assembly List

This is an integer block containing all of the information which relates the user's local degree of freedom numbers to the user's global numbering system. The master assembly list is, in effect, a Boolean logical transformation between the element displacement vectors, which together form a linearly dependent set, and the global displacement vector, which is an independent set. Entries in the master assembly list are referred to by:

INTGR(I) where $IMASTR \leq I \leq LMASTR$

The master assembly list need not be filled, but if it is not, a zero must be stored immediately following the last active position.

The master assembly list is subdivided into two sections, as indicated in Figure 2.

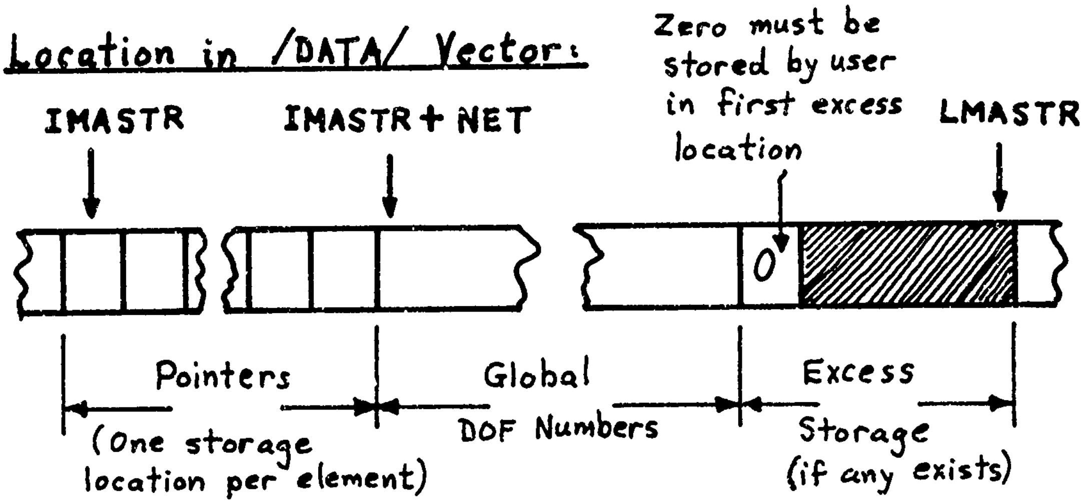

Location in /DATA/ Vector:
IMASTR
IMASTR + NET
Zero must be stored by user in first excess location
LMASTR
0
Pointers
(One storage location per element)
Global DOF Numbers
Excess Storage (if any exists)

FIGURE 2

The pointer section consists of one location per element, from INTGR(IMASTR) to INTGR(IMASTR+NET-1), with the convention that these locations correspond to the user's elements in ascending order 1, 2,..., NET. Each pointer contains the absolute address in the /DATA/ vector of the location where the assembly list for an element starts, as indicated in Figure 3.

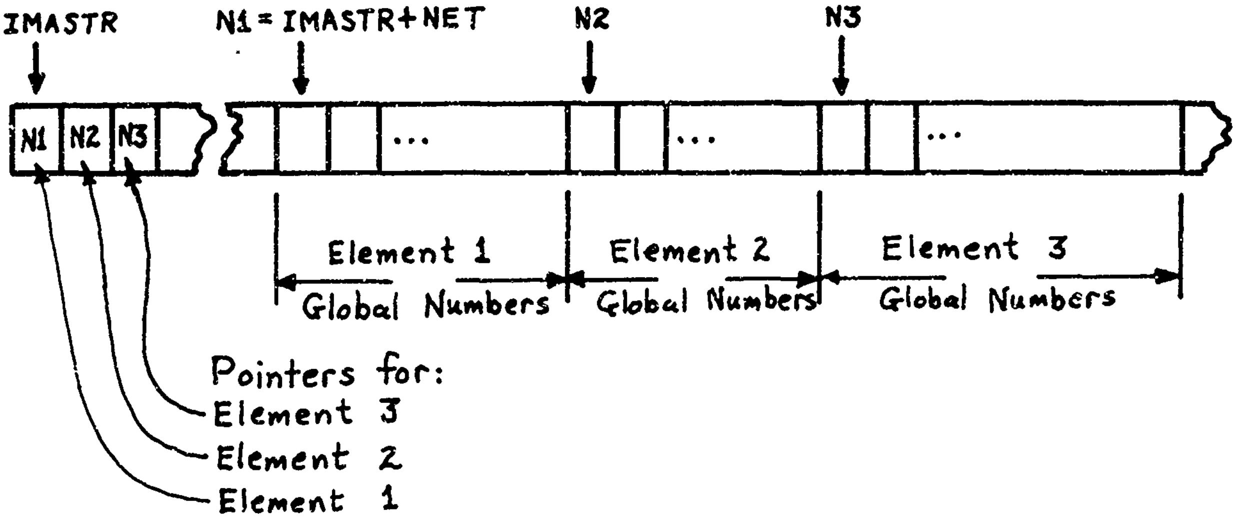


FIGURE 3

The remainder of the master list contains the user's element-by-element sequence of global displacement numbers. As an illustrative example, consider the set of elements shown in Figure 4, consisting of two 4-node rectangles and two 3-node triangles, each with two degrees of freedom per node. The total length required for the master assembly list is given by:

Pointers:	1 per element-----------	4
Rectangles:	8 DOF/element----------	16
Triangles:	6 DOF/element----------	12
TOTAL----------------------------------		32

Suppose that IMASTR=101. Then the assembly information, correctly stored, would appear as shown in Figure 5.

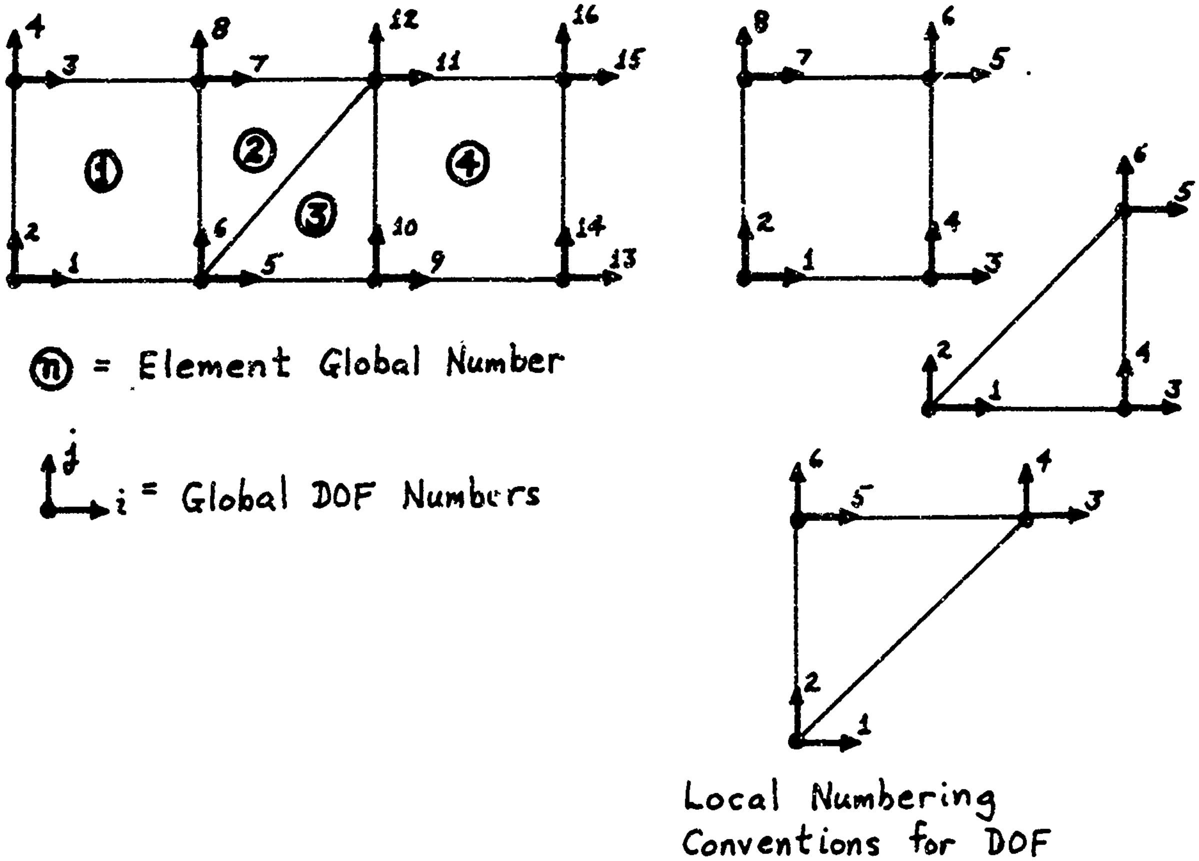
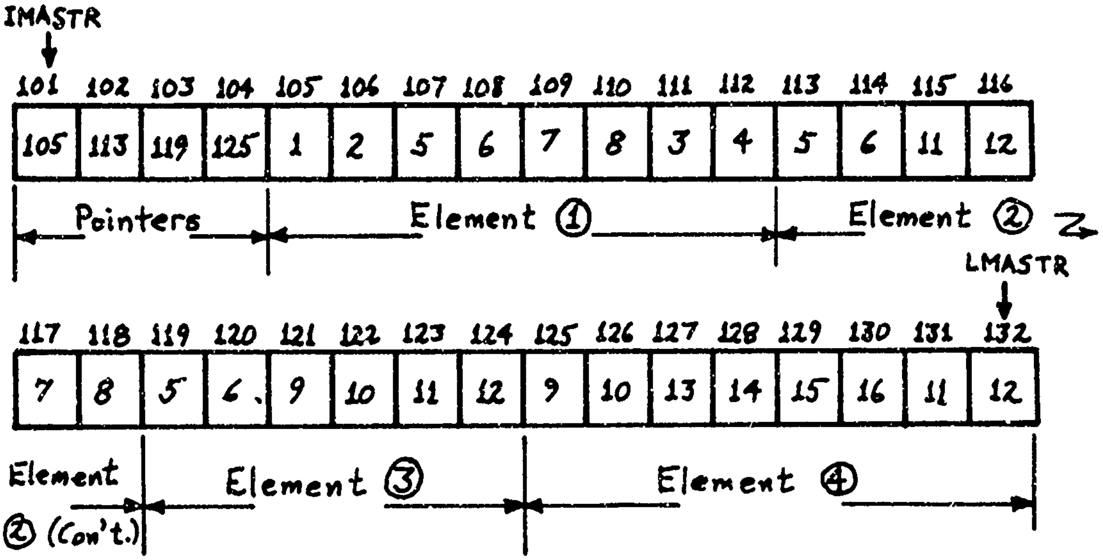


FIGURE 4



101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116
105	113	119	125	1	2	5	6	7	8	3	4	5	6	11	12

117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132
7	8	5	6	9	10	11	12	9	10	13	14	15	16	11	12

FIGURE 5

The following crude set of FØRTRAN instructions might be used to store the assembly information shown above:

```
NEXT=IMASTR+4
INTGR(IMASTR)=NEXT   (establishes pointer for 1st element)
INTGR(NEXT)=1
INTGR(NEXT+1)=2
⋮
INTGR(NEXT+7)=4
NEXT=NEXT+8
INTGR(IMASTR+1)=NEXT (establishes pointer for 2nd element)
INTGR(NEXT)=5
INTGR(NEXT+1)=6
⋮
INTGR(NEXT+5)=8
NEXT=NEXT+6
INTGR(IMASTR+2)=NEXT (establishes pointer for 3rd element)
:
```

The use of pointers in the master list allows assemblies involving as many different types of elements with different total numbers of degrees of freedom as desired. Also the use of DOF numbers rather than node numbers permits assembly of elements having different numbers of displacements at various nodes without requiring any special programming or conventions. The user may choose any element and displacement global numbering schemes and any local displacement numbering conventions he desires. The only restrictions are that:

1. There must be no gaps in the master assembly list.
2. The array must be filled or, if excess storage exists, a zero must be stored after the last displacement number.
3. All element numbers and degree of freedom numbers must be positive.
4. The lowest element number and the lowest global degree of freedom number must each be unity.

2.2.5 Force/Displacement Vector

This is a floating point block which contains all of the force and displacement information required for analysis of a structure. Entries in the force/displacement vector are referred to by:

```
REAL(I) where IQ≤I≤LQ
```

The Nth entry in the vector is the force or displacement associated with the Nth global degree of freedom.

Various types of information are overlayed in the force/displacement vector. First, if the structure being analyzed is loaded by continuum body forces (e.g., gravity) or is in a thermal environment, the resulting element equivalent nodal forces must be assembled along with the element stiffness matrices. FEABL subroutine ASEMBL uses the force/displacement vector as a storage area for this purpose. Second, the user must introduce his global concentrated nodal forces and prescribed displacements into the force/displacement vector. Finally, the FEABL solution subprograms store the displacement solution in the force/displacment vector by overwriting the prescribed quantities.

The following simple algorithm enables the user to communicate with the Nth global degree of freedom:

```
NN=IQ+N-1
REAL(NN)=...
or
...=f(REAL(NN))
```

For example, suppose the structure in Figure 4 is to be given the boundary conditions shown in Figure 6. Global degrees of freedom 4 and 8 have concentrated forces A and B applied, respectively, while degrees 12 and 15 have displacements C and D prescribed, respectively. Displacements, 1, 2, 5, 6, 13, and 14 are prescribed to be zero. The following set of FORTRAN instructions specify these boundary conditions:

(a) Establishment of the constraint vector

```
INTGR(ICØN)=1
INTGR(ICØN+1)=2
INTGR(ICØN+2)=5
INTGR(ICØN+3)=6
INTGR(ICØN+4)=12
INTGR(ICØN+5)=13
INTGR(ICØN+6)=14
INTGR(ICØN+7)=15
```

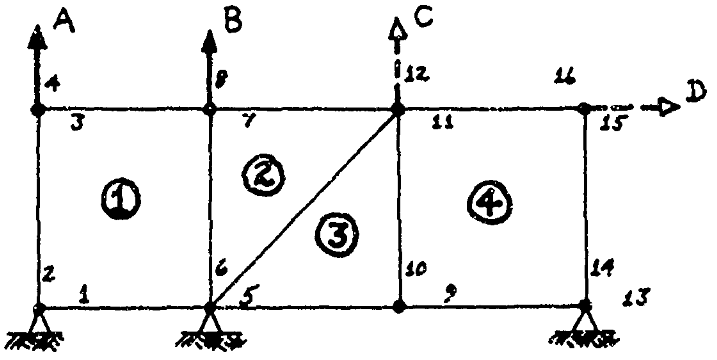
(b) Input of the prescribed displacements:

```
REAL(IQ)=0.
REAL(IQ+1)=0.
REAL(IQ+4)=0.
•
•
•
REAL(IQ+13)=0.
REAL(IQ+11)=C
REAL(IQ+14)=D
```

(c) Input of the prescribed forces:

```
REAL(IQ+3)=REAL(IQ+3)+A
REAL(IQ+7)=REAL(IQ+7)+B
```

The instructions (b) and (c) have been written assuming that element equivalent nodal forces have been assembled into the force/displacement vector. Thus, all degrees at which displacements are prescribed must be set to their correct values, while nonzero concentrated global forces must be added to the preexisting assembled element forces.



A, B are global concentrated forces

C, D are global prescribed displacements

FIGURE 6

2.2.6 Master Stiffness Matrix

This floating point block contains the essential entries of the master stiffness matrix, stored one row after the other. Entries in the master stiffness matrix are referred to by:

REAL(I) where $IK \leq I \leq LK$

Normally, the user is not required to communicate directly with the master stiffness matrix, but an understanding of its detailed organization will be helpful in debugging.

Since a stiffness matrix is always symmetric, only its lower triangle need be kept in storage. Thus, the general organization of the master stiffness array in the /DATA/ vector may be represented by the diagram shown in Figure 7.

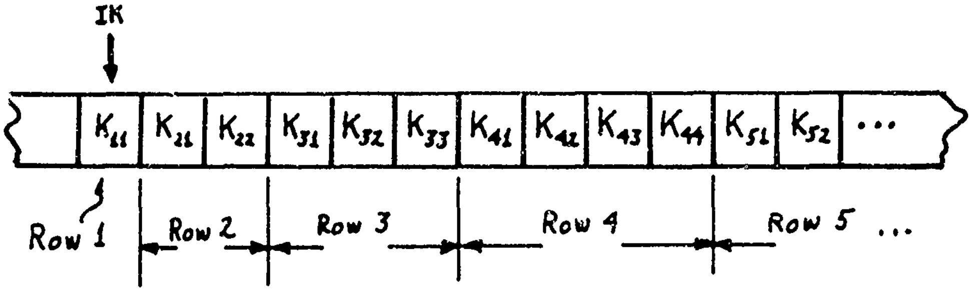


FIGURE 7

Significant additional savings in storage may be realized for problems with many degrees of freedom by taking advantage of the fact that a master stiffness matrix is normally banded and sparsely populated. The boundary line of the shaded area in Figure 8 represents the leading non-zero entry locations in a hypothetical stiffness matrix. FEABL subprograms which operate on the master stiffness matrix incorporate logic instructions which cause the operation to be skipped if the entry K_{ij} lies in the unshaded area of Figure 8. Thus, the leading zero entries for each row are not stored, and this is where the address count vector and the LNZ vector come into play. The actual organization of the master stiffness array consists of a sequence of

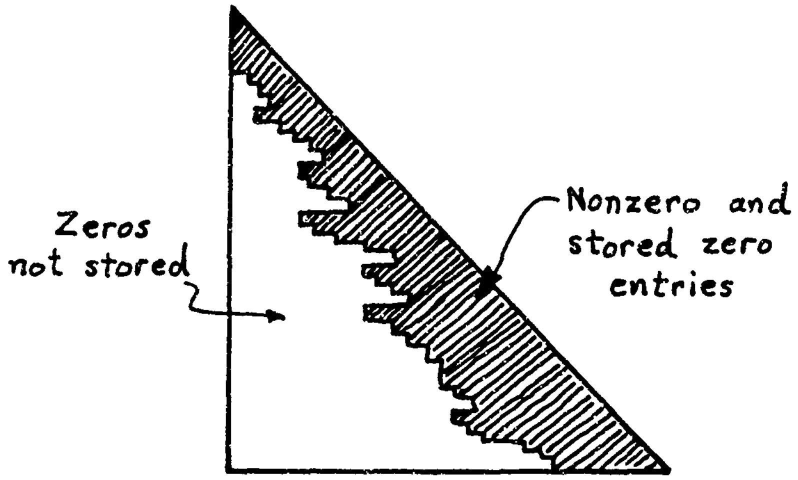


FIGURE 8

variable-length sections for the rows of the stiffness matrix. Figure 9 illustrates a sample sequence.

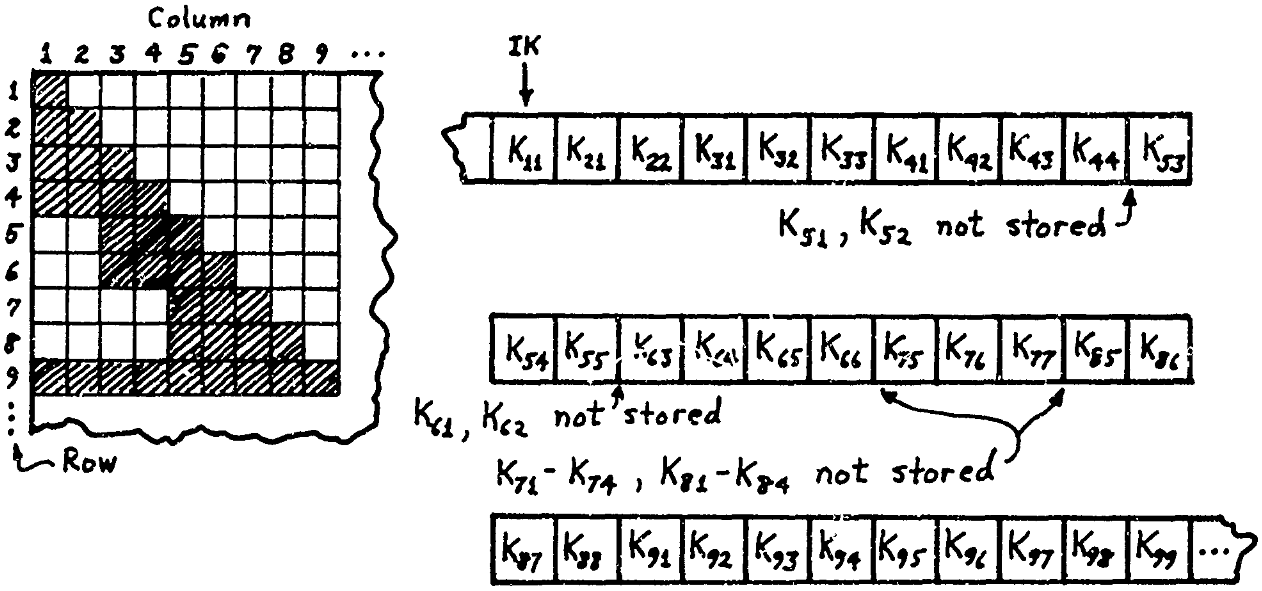


FIGURE 9

The algorithms for operating on the master stiffness matrix, based on the contents of the address count and LNZ vectors are quite simple:

```
     (For an operation on $K_{ij}$:  row I, column J, $J \leq I$)
     .
     .
     INDEX=ILNZ+I-1
     IF (J .LT. INTGR(INDEX)) GØ TØ 5
     KADR=IKOUNT+I-1
     KADR=INTGR(KADR)+J
     (To define address of $K_{ij}$ in master stiffness array)
     (Operate using REAL(KADR) for $K_{ij}$)
     .
     .
   5 (Skip operation if J<LNZ column number for the row)
```

2.3 Modification of the FEABL Data Storage System

The data storage system presented above has been designed for "production" computing. It has been assumed implicitly that the user will be conducting a great number of studies involving similar stress analysis problems and employing nearly the same number of degrees of freedom. Thus, the /DATA/ vector need be DIMENSIØNed only once, after which production object decks of the FEABL software may be made.

However, the FEABL software may also be placed in on-line storage in a form which will handle problems of widely varying size, with only minor modifications. These modifications are as follows:

1. Delete the CØMMØN /DATA/ REAL declaration from all programs and subprograms.
2. Delete the EQUIVALENCE (REAL(1), INTGR(1)) declaration from all subprograms. (Retain this declaration in MAIN.)
3. Use the standard declaration:

   DIMENSIØN REAL(2), INTGR(2)

   in all FEABL subroutines. (DIMENSIØN the /DATA/ vector properly in MAIN.)
4. Add the array names REAL and INTGR as arguments of

all FEABL subroutines, e.g.:

```
    SUBRØUTINE ØRK(LENGTH, REAL, INTGR)
    SUBRØUTINE FACTPD(REAL, INTGR)
```

When the on-line version of FEABL is used, only the DIMENSIØN declaration for the /DATA/ vector in the user's MAIN program need be changed to perform analyses requiring different amounts of data storage.

SECTION 3

FEABL PROCESS CONTROL

3.1 General

For present purposes the finite element analysis of a structure will be divided into eight programming stages:

1. Establishment of input/output device codes, problem size, and address index parameters.
2. Input of the master assembly list and organization of the master stiffness matrix into corresponding segments.
3. Generation and assembly of element stiffnesses (and nodal equivalent forces, if any) in global coordinates.
4. Application of rotation transformations to the master stiffness matrix (and assembled nodal equivalent forces) at any nodes at which the boundary conditions are to be given in special coordinate systems.
5. Input of the prescribed quantities, i.e., global numbers at which displacements are to be prescribed, values of prescribed displacements, and accumulation of values of any nonzero global concentrated forces. Application of constraints to the master stiffness matrix and force/displacement vector.
6. Solution for the master displacement vector.
7. Application of inverse rotation transformations to the master displacement vector, at any nodes where a rotation was applied in stage 4, to produce a master displacement vector entirely in the global coordinate system.
8. Extraction of element displacement vectors from the global vector; calculation of element stresses from the element displacement vector.

Specific FEABL subprograms are associated with each of the above stages, according to the following table:

Stage	FEABL Subroutines
1	SETUP
2	ØRK
3	ASEMBL
4	RØTATE
5	BCØN
6	FACTPD/FACTSD, SIMULQ
7	RØTATE
8	XTRACT

Figure 10 illustrates the standard FEABL process sequence in terms of the eight stages described above.

Analysts who are just beginning to work with FEABL are advised to observe strictly the standard process sequence outlined above. This sequence will be described in full detail in Section 4. More experienced analysts may find it convenient to depart occasionally from the standard process in order to reduce program length.

3.2 Description of the FEABL Software

The FEABL package consists of the eight subroutines listed in the table in Subsection 3.1. Each subroutine is associated with a particular stage in the standard process sequence (except RØTATE, which is associated with both stages 4 and 7). All FEABL subprograms are ready to use for in-core finite element analysis, once the DIMENSIØN declaration for the /DATA/ vector has been inserted (see Subsections 2.2 and 2.3). The following subsections demonstrate how each of the FEABL subroutines is called, define the subroutine arguments, and outline briefly what the subroutine does. A summary table of the CØMMØN area information required by each FEABL subroutine is given in Appendix A. Listings appear in Appendix C.

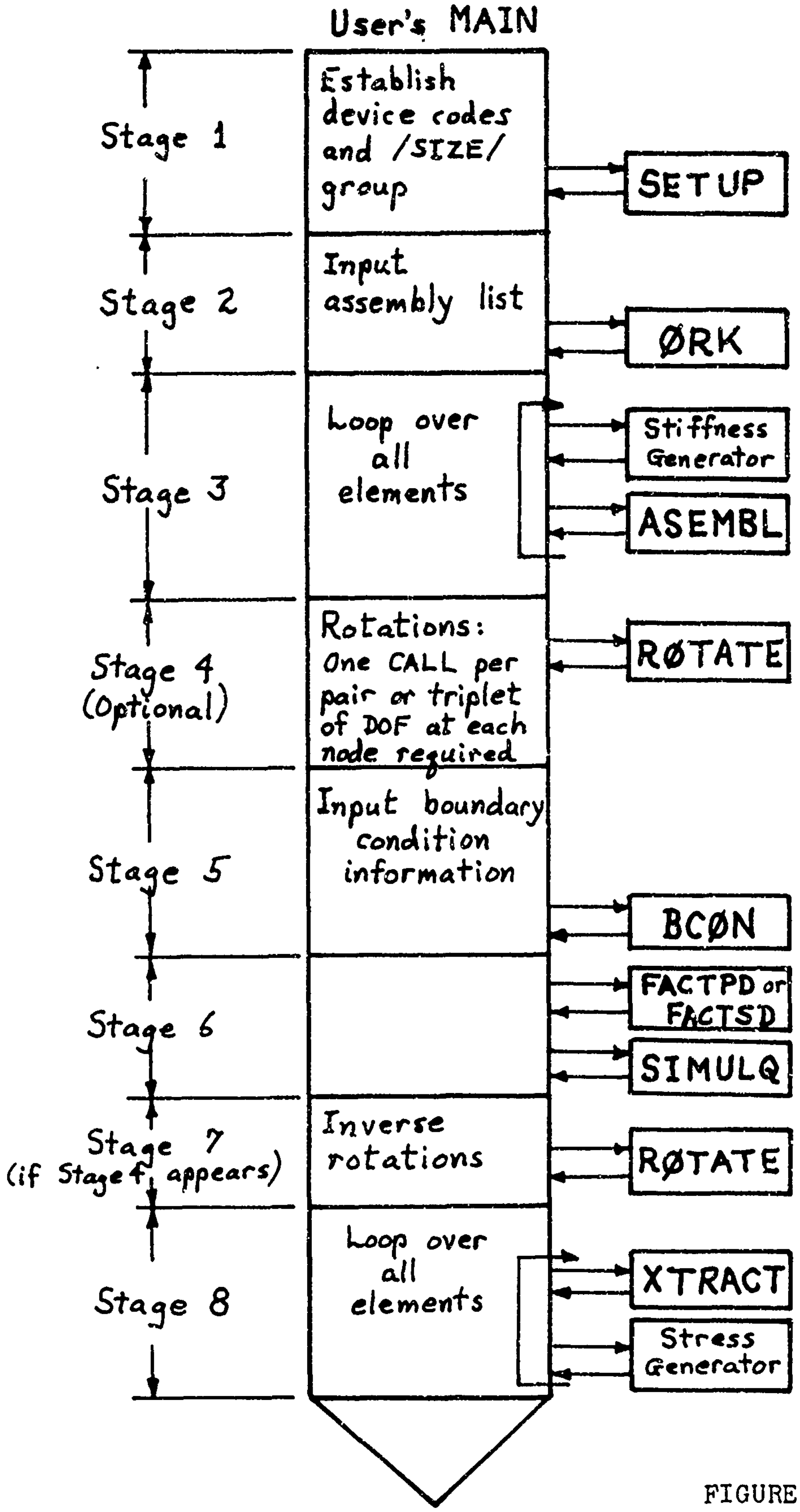

User's MAIN
Stage 1
Establish device codes and /SIZE/ group
SETUP
Stage 2
Input assembly list
ØRK
Stage 3
Loop over all elements
Stiffness Generator
ASEMBL
Stage 4 (Optional)
Rotations: One CALL per pair or triplet of DOF at each node required
RØTATE
Stage 5
Input boundary condition information
BCØN
Stage 6
FACTPD or FACTSD
SIMULQ
Stage 7 (if Stage 4 appears)
Inverse rotations
RØTATE
Stage 8
Loop over all elements
XTRACT
Stress Generator

FIGURE 10

3.2.1 Housekeeping Setup Subroutine for /DATA/ Vector (SETUP)

CALL SETUP(LENGTH, NCØN, MASTRL)

LENGTH - A scalar integer numerically equal to the dimension which the user has assigned to the /DATA/ vector.

NCON - A scalar integer greater than or equal to the total number of degrees of freedom, in the assembled structure, at which displacements are to be prescribed.

MASTRL - A scalar integer greater than or equal to the total number of words required for the master assembly list.

Based on NCØN, MASTRL and the total number of degrees of freedom in the entire assembled structure (NDT, in the /SIZE/ group), subroutine SETUP organizes the /DATA/ vector by calculating the address index parameters in the /BEGIN/ and /END/ groups, except for the index LK which defines the end of the block reserved for the master stiffness matrix. SETUP uses the argument LENGTH to test whether the user's /DATA/ vector has at least enough storage available to accommodate the first four data blocks (constraint vector, address count vector, LNZ vector and master assembly list). If the /DATA/ vector is too short, SETUP estimates the total length required for all six data blocks, based on a reasonable population density for the lower triangle of the master stiffness matrix, prints the estimate and aborts the run. If the first four blocks can be accommodated, the constraint vector is filled with zeros and control is returned to MAIN.

3.2.2 Subroutine for Detailed Organization of the Master Stiffness Matrix Block (ØRK)

CALL ORK(LENGTH)

LENGTH - A scalar integer numerically equal to the dimension which the user has assigned to the /DATA/ vector.

Subroutine ØRK produces, in essence, a map for the master stiffness matrix like that of Figure 9, using the information contained in the master assembly list to calculate the correct values of the entries in the LNZ vector. This is accomplished by first setting each LNZ column number equal to its row number (diagonal matrix), and then examining the assembly information element by element to re-set the LNZ column numbers, according to the following algorithm:

1. The smallest global number N for the element is found.
2. The element's global numbers are then treated as row numbers. If the LNZ column number corresponding to a row (global number) is greater than N, its value is re-set to N.

Once the LNZ vector has been established, subroutine ØRK uses an accumulation process to calculate the absolute address of the diagonal entry of each row. By convention, the diagonal is the only entry stored for the first row and is therefore located by the address index parameter IK:

```
INTGR(IKØUNT)=IK
```

Subsequent entries are located by the algorithm:

```
INTGR(IKØUNT+M-1)=INTGR(IKØUNT+M-2)+M+1-INTGR(ILNZ+M-1)
(Diag Addr for Mth row)=(Diag Addr for row M-1)+(Total no.
                                  of nonzero entries in Mth row)
```

Conveniently, the address of the diagonal for the last row in the master stiffness matrix is also the correct value of LK. Subroutine ØRK now tests the /DATA/ vector by means of the argument LENGTH. If the /DATA/ vector is too short for the problem data, an exact calculation of its required length is output and the run is aborted. If sufficient storage is available, the stiffness matrix map is output and the address count vector is modified by:

```
        INTGR(IKØUNT+M-1)=INTGR(IKØUNT+M-1)-M; 1≤M≤NDT
```

for all rows, M. This saves repeated subtraction of the row number in later subprograms. The correct algorithm for locating K_{MN} in the /DATA/ vector is now:

```
        KADR=INTGR(IKØUNT+M-1)+N; M≥N
```

K_{MN} is assigned to REAL(KADR)

Since the next stage of the analysis will involve the accumulation of data in the master stiffness matrix (and perhaps in the force/displacement vector), ØRK's last action before returning control to MAIN is to fill these two data blocks with floating point zeros.

3.2.3 Element Assembly Subroutine (ASEMBL)

CALL ASEMBL(LNUM, NDE, ELK, ELQ)

LNUM - A positive scalar integer = user's global element number

NDE - A scalar integer = total number of degrees of freedom possessed by the element which is about to be assembled

ELK - A floating point, two-dimensional square array which contains the stiffness matrix of the element about to be assembled.

ELQ - A floating point vector which contains the equivalent nodal forces for the element about to be assembled, or which contains floating point zeros if there are no equivalent nodal forces.

Since ELK and ELQ are variably DIMENSIØNed in this subprogram, elements having different numbers of degrees of freedom can be handled automatically. However, these arguments must be DIMENSIØNed explicitly in the user's MAIN program. For example, suppose a structure is to be analyzed in plane stress with a combination of 3-node triangle elements (6 degrees of freedom) and 4-node rectangle elements (8 degrees of freedom). Then the declaration:

```
        DIMENSIØN TRIK(6,6), TRIQ(6), RECK(8,8), RECQ(8)
```

might appear in MAIN. A triangle element would be assembled by the instruction:

```
        CALL ASEMBL(LNUM, 6, TRIK, TRIQ)
```

while a rectangle element would be assembled by:

```
        CALL ASEMBL (LNUM, 8, RECK, RECQ)
```

Subroutine ASEMBL can handle elements having as many as one hundred degrees of freedom (not a serious restriction). If the assembly of larger elements is attempted, ASEMBL will abort the run and tell the user to change the DIMENSIØN of one of its internal parameters.

ASEMBL examines the section of the master assembly list belonging to element number LNUM and records the values of the global displacement numbers in an internal vector called MNUM. Then, each entry K_{ij} of the lower triangle of the element stiffness matrix ($i \geq j$), is accumulated into its proper place in the master stiffness matrix, according to the algorithm:

```
        I=MNUM(i)
        J=MNUM(j)
        $K_{ij} \to K_{IJ}$ for $I \geq J$
        $K_{ij} \to K_{JI}$ for $I < J$
```

(See Subsection 3.2.1 for address algorithm for K_{IJ}.) ASEMBL does not use the upper triangle ($i<j$) of the element stiffness matrix; the user may omit calculating these entries to save time. The vector ELQ of element equivalent nodal forces is accumulated in the same manner into the proper locations in the master force vector (fifth block of the /DATA/ vector).

3.2.4 Rotation Transformation Subroutine (RØTATE)

CALL RØTATE(NØDE, IRØW, JRØW, KRØW, ZANGLE, YANGLE, XANGLE)

NØDE - A scalar integer "convenience" number for the user. May be positive or negative but not zero. (See explanation below.)

IRØW }
JRØW } Three scalar integers equal to the global numbers of the degrees of freedom at the node at which the rotation is to be performed.
KRØW }

ZANGLE }
YANGLE } Floating point values of the three Euler rotation angles in degree measure.
XANGLE }

The argument NØDE does not enter directly into the transformation calculations, but is printed out in the heading of the information supplied by RØTATE. Normally, the user assigns either his node numbers or the series 1, 2, 3,... to a set of rotations. A positive value of NØDE causes the full subprogram to execute, and is used for stage 4 in the process sequence. In stage 7, only the solution displacement vector requires transformation; a negative value of NØDE will cause execution of the now unwanted transformation of the stiffness matrix to be skipped.

The global numbers of the degrees of freedom at the node being rotated may be contiguous (e.g., 13, 14, 15) or separated (e.g., 5, 10, 15) depending upon the global numbering scheme the user has adopted. Either form is acceptable to this subprogram. If only two degrees of freedom are to participate in the rotation (as, for example, in plane stress problems), an integer zero must be specified for the third global number. If six degrees of freedom are to participate (e.g., shell elements), two separate rotations are required, i.e., one CALL for the translational and one for the rotational degrees of freedom. Errors in the global number arguments (e.g., repeating a number or too many zeros) will cause an abort.

The three Euler angle arguments must be specified according to the conventions illustrated in Figure 11. Let XYZ be the user's global cartesian axis system, with respect to which the element stiffness and equivalent nodal forces have been generated and assembled. Let $\tilde{X}\tilde{Y}\tilde{Z}$ by a local coordinate system with respect to which the displacement constraints are to be specified.

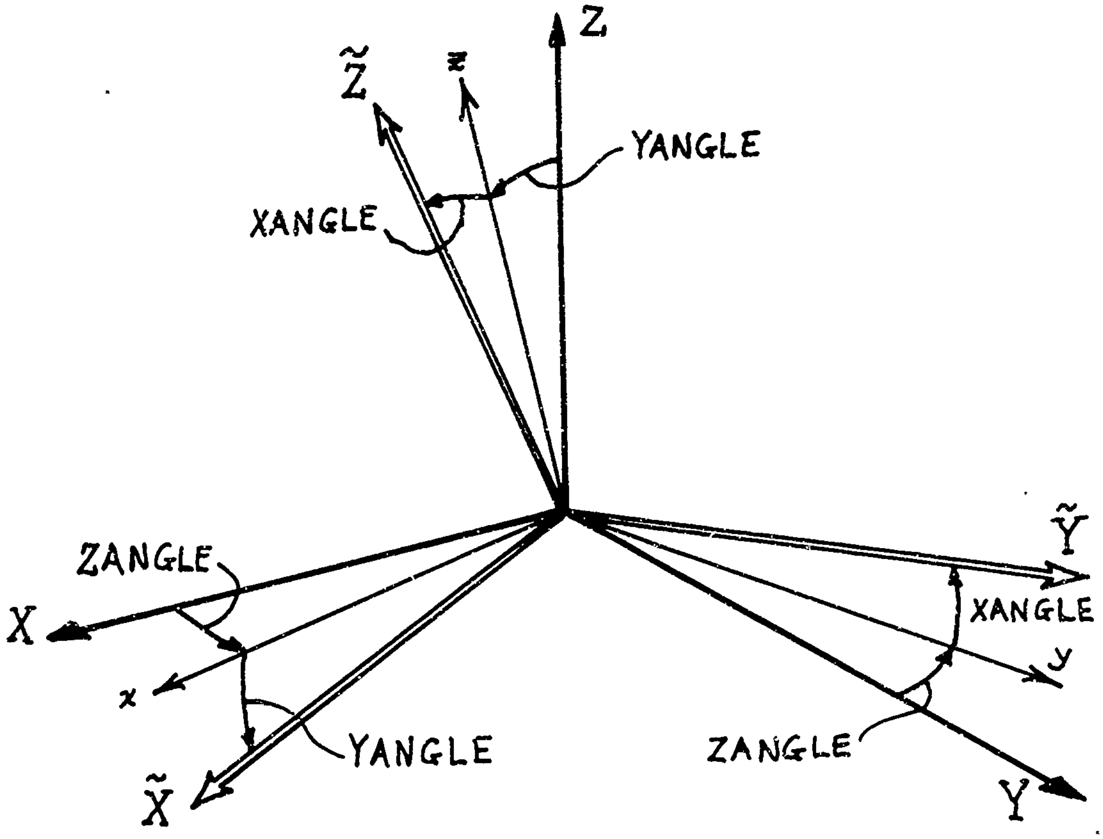


FIGURE 11

The following conventions have been adopted in subroutine RØTATE:

1. Axes X, Y are first rotated through angle ZANGLE about axis Z to the intermediate positions x, y.
2. Axes x, Z are then rotated through angle YANGLE about axis y, x to the final position $\tilde{X}$ and Z to an intermediate position z.
3. Axes y, z are then rotated through angle XANGLE about axis $\tilde{X}$ to their final positions $\tilde{Y}$, $\tilde{Z}$.
4. Positive angles obey the "right hand rule" of vector analysis.

Subroutine ROTATE forms the matrix of direction cosines:

$$\mathbf{D} = \begin{bmatrix} \cos(\tilde{X}, X) & \cos(\tilde{X}, Y) & \cos(\tilde{X}, Z) \\ \cos(\tilde{Y}, X) & \cos(\tilde{Y}, Y) & \cos(\tilde{Y}, Z) \\ \cos(\tilde{Z}, X) & \cos(\tilde{Z}, Y) & \cos(\tilde{Z}, Z) \end{bmatrix}$$

from the values of the Euler angles supplied in the argument list. Then, if argument NØDE is positive, RØTATE applies the following transformations to portions of the master stiffness matrix $\mathbf{K}$ where i, j, k are the global numbers specified by the user:

$$\mathbf{D} \begin{bmatrix} K_{ii} & K_{ij} & K_{ik} \\ K_{ji} & K_{jj} & K_{jk} \\ K_{ki} & K_{kj} & K_{kk} \end{bmatrix} \mathbf{D}^T$$

$$\mathbf{D} \left[\begin{array}{lllll} K_{i1} & K_{i2} \cdots K_{i,i-1} & K_{i,i+1} \cdots K_{i,j-1} & K_{i,j+1} \cdots K_{i,k-1} & K_{i,k+1} \cdots \\ K_{j1} & K_{j2} \cdots K_{j,i-1} \cdots & & & \\ K_{k1} & K_{k2} \cdots & & & \end{array} \right.$$

$$\begin{bmatrix} K_{1i} & K_{1j} & K_{1k} \\ K_{2i} & K_{2j} & K_{2k} \\ \vdots & \vdots & \vdots \\ K_{i-1,i} & K_{i-1,j} & \\ & \vdots & \\ K_{i+1,i} & & \\ \vdots & & \\ K_{j-1,i} & & \\ K_{j+1,i} & & \\ \vdots & & \\ K_{k-1,i} & & \\ K_{k+1,i} & & \\ \vdots & & \end{bmatrix} \mathbf{D}^T$$

Finally, the corresponding entries of the force/displacement vector are transformed according to:

$$\mathbf{D} \begin{Bmatrix} Q_i \\ Q_j \\ Q_k \end{Bmatrix}$$

A few examples will illustrate the proper use of subroutine RØTATE. First, suppose that global degrees of freedom 1 and 2 are to be rotated by 45 degrees at node 1 in a plane elasticity problem. The correct instruction is:

```
        CALL RØTATE(1, 1, 2, 0, 45., 0., 0.)
```

To return the force/displacement vector to the user's global axis system after the displacement solution has been obtained:

```
        CALL RØTATE(-1, 1, 2, 0, -45., 0., 0.)
```

Note that only the first angular argument ZANGLE is used in two-dimensional problems. In a three-dimensional problem, to rotate degrees 4, 5, 6 at node 2 through angles of 15, 30 and 45 degrees about the Z, y and $\tilde{X}$ axes:

```
        CALL RØTATE(2, 4, 5, 6, 15., 30., 45.)
```

and after the displacement solution has been obtained:

```
        CALL RØTATE(-2, 4, 5, 6, -45., -30., -15.)
```

Note that not only the signs, but also the order of application of the angles is reversed. However, some care is required in three dimensions. Suppose degrees 7, 8, 9 at node 3 were rotated only about axes Z and y:

```
        CALL RØTATE(3, 7, 8, 9, 20., 40., 0.)
```

Then to reverse the rotation after obtaining the displacement solution:

```
        CALL RØTATE(-3, 7, 8, 9, -40., -20., 0.)
```

Note that the unused XANGLE does not participate in the reversal. The instruction:

```
        CALL RØTATE(-3, 7, 8, 9, 0., -40., -20.)
```

would be incorrect.

3.2.5 Boundary Constraint Subroutine (BCØN)

```
        CALL BCØN
```

Let $\boldsymbol{u}$, $\boldsymbol{Q}$ represent respectively the global* displacement vector and the global force vector. The result of the first four program stages has been to supply a right-hand side and the stiffness coefficients K_{ij} for the force-displacement relations:

$$\mathbf{K}\boldsymbol{u} = \boldsymbol{Q}$$

However, the value of u is known at some degrees of freedom, with the corresponding Q unknown, while the value of Q is known

*The term "global" should be taken in a more general sense here. It refers to the final set of coordinate systems $\tilde{X}\tilde{Y}\tilde{Z}$ in which the user will prescribe his boundary conditions.

at the other degrees.

Conceptually, the set of all degrees of freedom in the structure may be divided into two subsets: Those at which forces are prescribed (F) and those at which displacements are prescribed (D). When the user specifies the values of the prescribed quantities, $\boldsymbol{Q}$ becomes a "force/displacement" vector in fact:

$$\boldsymbol{Q} = \begin{Bmatrix} \hat{Q}_F \\ \hat{u}_D \end{Bmatrix}$$ ("^" means a prescribed quantity)

and:

$\boldsymbol{u}_F$ = unknown displacements
$\boldsymbol{Q}_D$ = unknown reaction forces

The force-displacement relations may be partitioned in a similar manner:

$$\begin{bmatrix} \mathbf{K}_{FF} & \mathbf{K}_{FD} \\ \mathbf{K}_{FD}^T & \mathbf{K}_{DD} \end{bmatrix} \begin{Bmatrix} u_F \\ \hat{u}_D \end{Bmatrix} = \begin{Bmatrix} \hat{Q}_F \\ Q_D \end{Bmatrix}$$

Subroutine BCØN transforms the force-displacement relations from the above form to:

$$\begin{bmatrix} \mathbf{K}_{FF} & 0 \\ 0 & \mathbf{I} \end{bmatrix} \begin{Bmatrix} u_F \\ \hat{u}_D \end{Bmatrix} = \begin{Bmatrix} \hat{Q}_F - \mathbf{K}_{FD}\hat{u}_D \\ \hat{u}_D \end{Bmatrix}$$

With the above "right hand side" in the force/displacement vector, standard equation-solving techniques may be used to produce the solution displacement vector. Before making the transformation, BCØN arranges the entries of the constraint vector (Subsection 2.2.1) in ascending order and checks for the presence of global numbers (positive integers). The run is aborted if no global numbers are found.

3.2.6 Stiffness Matrix Factoring Subroutine (FACTPD/FACTSD)

CALL FACTPD

or

CALL FACTSD

The force-displacement relations are solved by Choleski's direct method, which consists of two programming steps:

1. The master stiffness matrix is factored into a triple product.
2. The displacements are solved for sequentially, in three sub-steps.

Subroutine FACTPD/FACTSD accomplishes the first programming step. The master stiffness matrix $\mathbf{K}$ is factored into the form:

$$\mathbf{K} = \mathbf{L}\mathbf{D}\mathbf{L}^T$$

where $\mathbf{L}$ is a lower triangular matrix:

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 & 0\ldots \\ L_{21} & 1 & 0 & 0\ldots \\ L_{31} & L_{32} & 1 & 0\ldots \\ L_{41} & L_{42} & L_{43} & 1\ldots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

and $\mathbf{D} = \lceil D_1 \; D_2 \ldots \rfloor$ is a diagonal matrix. The factoring algorithms are:

$$\left.\begin{array}{l} L_{mn} = \frac{1}{D_n}\left[K_{mn} - \sum_{j=J(m,n)}^{n-1} (D_j L_{mj} L_{nj})\right] \ ; \ n = J(n), \ldots, m-1 \\ \qquad D_m = K_{mm} - \sum_{j=J(m)}^{m-1} (D_j L_{mj}^2) \end{array}\right\} m = 1, 2, \ldots, NDT$$

where:

$J(n)$ = Leading Nonzero Entry Column Number for Row n
$J(m)$ = Leading Nonzero Entry Column Number for Row m
$J(m,n) = \max(J(n), J(m))$

The master stiffness matrix is destroyed and replaced by the entries of $\mathbf{L}$ and $\mathbf{D}$ as the factoring process is executed. The entries L_{mn} $(m>n)$ and D_m are stored respectively at K_{mn} $(m>n)$ and K_{mm}. The unit diagonal entries of $\mathbf{L}$ are not stored.

FACTPD/FACTSD tests the entries of $\mathbf{D}$ for nonsingularity and positive-definiteness as they are created. $\mathbf{K}$ is positive-definite if all $D_m>0$. Rows for which $D_m<0$ are reported. If any D_m is found to equal zero exactly, $\mathbf{K}$ is singular; the row in which the singularity was discovered is reported and the run is aborted.

The names FACTPD and FACTSD refer to different entry points in this subprogram. If $\mathbf{K}$ is supposed to be positive-definite (as in the case of a structure analyzed by compatible displacement elements), FACTPD should be called. If errors were made in the assembly or constraint stages of the program, they will appear now as $D_m<0$ in one or more rows, and FACTPD will abort the run. If $\mathbf{K}$ is not necessarily positive-definite (as in the case of a structure analyzed by hybrid stress-displacement elements), FACTSD should be called. FACTSD continues execution even if there exist $D_m<0$. If the FEABL software is to be converted to the on-line storage version (Section 2.3), be sure to make the array names REAL, INTGR arguments of <u>both</u> entry points.

FACTPD/FACTSD also makes a rough estimate of the conditioning of $\mathbf{K}$ by calculating the so-called rounding error parameter (see Ref. 2, pg. 81):

$$E = \min \left(|D_m| / |K_{mm}| \right)$$

E is a measure of how many significant figures of information have been lost in the diagonal entries, as a result of the factoring algorithm. The run is aborted if $E<10^{-5}$. The user must keep in mind the fact that the rounding error parameter is an imperfect conditioning measure. If FACTPD/FACTSD reports that no significant figures have been lost, it does not necessarily follow that there is no error in the displacement solution. Advanced techniques for realistic solution error estimates are discussed in Section 5.

3.2.7 Subroutine for Solution of Simultaneous Equations (SIMULQ)

CALL SIMULQ(ENERGY)

ENERGY - A scalar floating point variable, the value of which is undefined when SIMULQ is called.

With the stiffness matrix in factored form $\mathbf{K} = \mathbf{L}\mathbf{D}\mathbf{L}^T$, let $\mathbf{P} = \mathbf{L}^T \boldsymbol{u}$ and $\mathbf{R} = \mathbf{D}\mathbf{P}$. Then:

$$\mathbf{L}\mathbf{R} = \mathbf{Q}$$

can be solved sequentially for R_1, R_2,..., R_{NDT} via the algorithm:

$R_1 = Q_1$

$$R_m = Q_m - \sum_{j=J(m)}^{m-1} \left(L_{mj} R_j \right) \quad ; \quad m = 2,3, \ldots, NDT$$

Then:

$$P_m = R_m / D_m \quad ; \quad m = 1,2, \ldots, NDT$$

Finally, the displacement solution is obtained by solving $\mathbf{L}^T \boldsymbol{u} = \mathbf{P}$ sequentially for u_{NDT}, $u_{NDT-1}, \ldots, u_1$ according to:

$$u_{NDT} = P_{NDT}$$

$$u_m = P_m - \sum_{j=m+1}^{NDT} (L_{jm} u_j) \; ; \; m = NDT-1, NDT-2, \ldots, 1$$

As SIMULQ carries out these three sub-steps, the prescribed vector $\boldsymbol{Q}$ is first replaced by $\mathbf{R}$, then $\mathbf{R}$ is replaced by $\mathbf{P}$ and finally $\mathbf{P}$ is replaced by $\boldsymbol{u}$. $\boldsymbol{Q}$ and $\boldsymbol{u}$ are printed out by SIMULQ.

Since the approximate value of the strain energy in the structure is often useful to the analyst, this quantity is calculated by SIMULQ during execution of the solution steps. If the factored form of $\mathbf{K}$ is introduced into the strain energy expression, there results:

$$\text{Strain Energy} = \frac{1}{2} \boldsymbol{u}^T \mathbf{K} \boldsymbol{u} = \frac{1}{2} (\boldsymbol{u}^T \mathbf{L}) \mathbf{D} (\mathbf{L} \boldsymbol{u}) = \frac{1}{2} \mathbf{P}^T \mathbf{D} \mathbf{P}$$

Thus, the straightforward algorithm:

$$\text{Strain Energy} = \frac{1}{2} \sum_{j=1}^{NDT} D_j (P_j)^2$$

can be used. The value of the strain energy thus calculated is printed out. At the end of execution of subroutine SIMULQ, the force/displacement vector (fifth block in the /DATA/ vector) contains the master displacement solution vector, and the strain energy value has been assigned to the argument ENERGY.

3.2.8 Subroutine for Extraction of Element Displacements from the Global Displacement Vector (XTRACT)

CALL XTRACT(LNUM, NDE, ELQ)

LNUM - A positive scalar integer equal to user's global element number

NDE - A scalar integer equal to the total number of degrees of freedom in the element

ELQ - A floating point vector at least NDE words long. The entries of ELQ are undefined when XTRACT is called.

If the analyst desires to calculate stress or strain distributions in his structure, he commonly uses transformations between stress or strain and nodal displacements:

$$\boldsymbol{\varepsilon} = \mathbf{B}\,\boldsymbol{u}_{el} \qquad \boldsymbol{\sigma} = \mathbf{E}\,\boldsymbol{\varepsilon}$$

where $\boldsymbol{\varepsilon}, \boldsymbol{\sigma}$ are respectively vectors of strain and stress components at selected points in the element domain, and where $\boldsymbol{u}_{el}$ is the vector of element nodal displacements, i.e., a subset of the global vector $\boldsymbol{u}$. Subroutine XTRACT selects the correct subset $\boldsymbol{u}_{el}$ out of $\boldsymbol{u}$, based upon the information contained in the user's master assembly list. The values of $\boldsymbol{u}_{el}$ are placed in the argument vector ELQ.

SECTION 4

A SAMPLE FEABL PROGRAM

In order to illustrate further the data location and process sequence interfacing, sample user programs will be developed for analysis of the truss structure shown in Figure 12. The structure consists of 16 bars and 18 degrees of freedom in the XY plane.

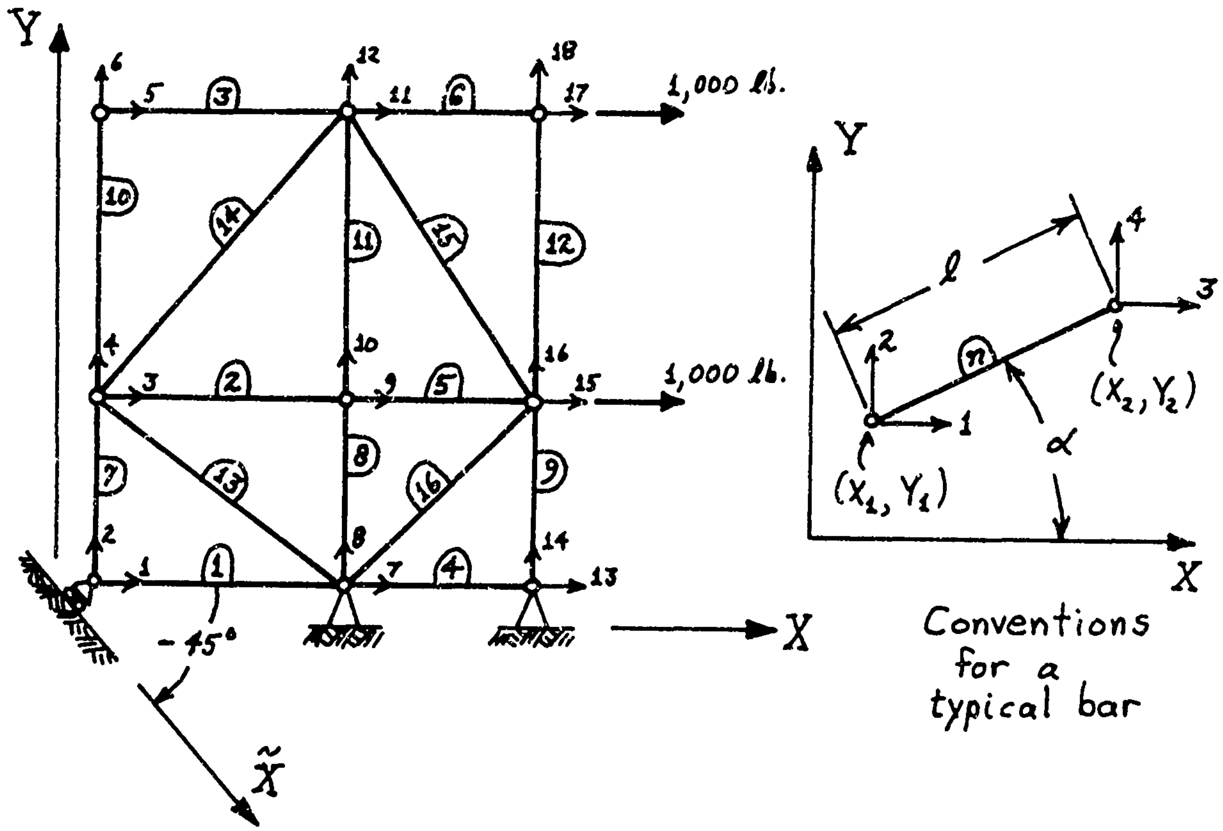


FIGURE 12

Local number conventions for the ⓝ th typical bar element are shown at the right. With these conventions, the stiffness matrix of the typical bar element is given by:

$$\mathbf{k}_{\textcircled{n}} = \frac{EA}{\ell}\begin{bmatrix} \cos^2\alpha & & & (\text{Symmetric}) \\ \sin\alpha\cos\alpha & \sin^2\alpha & & \\ -\cos^2\alpha & -\sin\alpha\cos\alpha & \cos^2\alpha & \\ -\sin\alpha\cos\alpha & -\sin^2\alpha & \sin\alpha\cos\alpha & \sin^2\alpha \end{bmatrix}$$

where:

$$\ell = \sqrt{(X_2-X_1)^2 + (Y_2-Y_1)^2}$$

$$\cos\alpha = \frac{1}{\ell}(X_2-X_1)$$

$$\sin\alpha = \frac{1}{\ell}(Y_2-Y_1)$$

and where E, A are the bar's modulus and cross section area. Also, once the displacements u_1, u_2, u_3, u_4 are known for the element, its elongation may be calculated as:

$$\delta = (u_3-u_1)\cos\alpha + (u_4-u_2)\sin\alpha$$

and the load in the bar is then given by:

$$P = EA\delta/\ell$$

where ℓ and α are defined as above.

The user decides to read the properties and nodal coordinates for each element, each time they are required (a rather inefficient procedure). Reading is to be done by the generator subroutine, rather than in MAIN; however, the card reader device code will be established in MAIN. Therefore, the user programs his stiffness matrix and stress generator subroutines as follows:

Stiffness Matrix Generator

```
      SUBRØUTINE BARK (ELK, ELQ)
      DIMENSIØN ELK (4, 4), ELQ (4)
      CØMMØN /IØ/ KR, KW, KP, KT1, KT2, KT3
C  INTERFACE WITH IØ CØNTRØL PARAMETERS IS ØPTIØNAL
   91 FØRMAT (6E10.3)
      READ (KR, 91) X1, Y1, X2, Y2, E, A
C  CALCULATE BAR LENGTH
      BARL=SQRT ((X2-X1)** 2+(Y2-Y1)** 2)
C  CALCULATE SINE AND CØSINE
      S=(Y2-Y1)/BARL
      C=(X2-X1)/BARL
C  CALCULATE ENTRIES IN LØWER TRIANGLE ØF ELK-ASEMBL DØES
C  NØT USE UPPER TRIANGLE
      ELK (1, 1)=E*A*C*C/BARL
      ELK (2, 1)=E*A*S*C/BARL
      .
      .
      etc.
      .
      ELK (4, 4)=E*A*S*S/BARL
C  ESTABLISH ZERØ ELEMENT EQUIVALENT NØDAL FØRCES
      DØ 10 I=1, 4
   10 ELQ (I)=0.
      RETURN
      END
```

Stress Generator

```
      SUBRØUTINE BARF (LNUM, ELQ)
      DIMENSIØN ELQ (4)
      CØMMØN /IØ/ KR, KW, KP, KT1, KT2, KT3
   91 FØRMAT (6E10.3)
   92 FØRMAT (21HOBAR FØRCE IN BAR NØ., I4, 2H =, E10.3, 3H LB)
      READ (KR, 91) X1, Y1, X2, Y2, E, A
      BARL = SQRT ((X2-X1)** 2+(Y2-Y1)**2)
      S=(Y2-Y1)/BARL
      C=(X2-X1)/BARL
      FORCE=E*A*(C*(ELQ(3) - ELQ (1)) + S* (ELQ(4)-ELQ (2)))/BARL
      WRITE (KW, 92) LNUM, FØRCE
      RETURN
      END
```

Subroutines BARK and BARF are ready to use in conjunction with the FEABL software. The user now begins the construction of his MAIN program, one stage at a time.

Stage 1: Program Heading (Data Location Interface), Device Code and Problem Size Establishment

The user estimates that a 1000-word /DATA/ vector will be more than adequate for the problem. Required device codes are the card reader (5) and printer (6).

```
C  MAIN PRØGRAM FØR SØLUTIØN ØF TRUSS PRØBLEM
     DIMENSIØN REAL(1000), INTGR(1000)
     DIMENSIØN ELK(4, 4), ELQ(4)
     CØMMØN /IØ/ KR, KW, KP, KT1, KT2, KT3
     CØMMØN /SIZE/ NET, NDT
     CØMMØN /BEGIN/ ICØN, IKØUNT, ILNZ, IMASTR, IQ, IK
     CØMMØN /END/ LCØN, LKØUNT, LLNZ, LMASTR, LQ, LK
     EQUIVALENCE (REAL(1), INTGR(1))
     KR=5
     KW=6
     NET=16
     NDT=18
     CALL SETUP (1000, 5, 80)
C  END ØF STAGE 1
```

The user has called for space for five constraints: master displacements 2 (after a -45° rotation of 1 and 2), 7, 8, 13 and 14. The assembly list must allow room for 16 elements x (4 DOF plus 1 pointer per element) = 80 words. The first declaration:

```
DIMENSIØN REAL(1000), INTGR(1000)
```

is also duplicated and placed in each FEABL subroutine.

Stage 2: Assembly List Input and Organization of **K**

The user recognizes that the pointers will occupy locations IMASTR to IMASTR+15 (see Section 2.2.4). He chooses to write specific assignment instructions for each element:

```
INTGR(IMASTR)=IMASTR+16 (Pointer for 1st element)
INTGR(IMASTR+16)=1
INTGR(IMASTR+17)=2
INTGR(IMASTR+18)=7
INTGR(IMASTR+19)=8
•
•
•
```

Stage 1: Program Heading (Data Location Interface), Device Code and Problem Size Establishment

The user estimates that a 1000-word /DATA/ vector will be more than adequate for the problem. Required device codes are the card reader (5) and printer (6).

```
C  MAIN PRØGRAM FØR SØLUTIØN ØF TRUSS PRØBLEM
     DIMENSIØN REAL(1000), INTGR(1000)
     DIMENSIØN ELK(4, 4), ELQ(4)
     CØMMØN /IØ/ KR, KW, KP, KT1, KT2, KT3
     CØMMØN /SIZE/ NET, NDT
     CØMMØN /BEGIN/ ICØN, IKØUNT, ILNZ, IMASTR, IQ, IK
     CØMMØN /END/ LCØN, LKØUNT, LLNZ, LMASTR, LQ, LK
     EQUIVALENCE (REAL(1), INTGR(1))
     KR=5
     KW=6
     NET=16
     NDT=18
     CALL SETUP (1000, 5, 80)
C  END ØF STAGE 1
```

The user has called for space for five constraints: master displacements 2 (after a -45° rotation of 1 and 2), 7, 8, 13 and 14. The assembly list must allow room for 16 elements x (4 DOF plus 1 pointer per element) = 80 words. The first declaration:

```
DIMENSIØN REAL(1000), INTGR(1000)
```

is also duplicated and placed in each FEABL subroutine.

Stage 2: Assembly List Input and Organization of **K**

The user recognizes that the pointers will occupy locations IMASTR to IMASTR+15 (see Section 2.2.4). He chooses to write specific assignment instructions for each element:

```
INTGR(IMASTR)=IMASTR+16 (Pointer for 1st element)
INTGR(IMASTR+16)=1
INTGR(IMASTR+17)=2
INTGR(IMASTR+18)=7
INTGR(IMASTR+19)=8
.
.
.
```

```
       •
       •
       •
       INTGR(IMASTR+7)=IMASTR+44 (pointer for 8th element)
       INTGR(IMASTR+44)=7
       INTGR(IMASTR+45)=8
       INTGR(IMASTR+46)=9
       INTGR(IMASTR+47)=10
       •
       •
       •
       INTGR(IMASTR+15)=IMASTR+76 (pointer for 16th element)
       INTGR(IMASTR+76)=7
       INTGR(IMASTR+77)=8
       INTGR(IMASTR+78)=15
       INTGR(IMASTR+79)=16 (80th location in master assembly list)
```

At this point, it is advisable to dump the assembly list for debugging purposes if there is any doubt about its accuracy. Dumping may be done by:

```
       WRITE (KW, 5) (INTGR(I), I=IMASTR, LMASTR)
  5    FØRMAT (1X, 10I10)
```

Finally:

```
       CALL ØRK(1000)
  C    END ØF STAGE 2
```

Stage 3: Generation and Assembly of Element Properties

In this stage, the user merely invokes the appropriate subroutines in a loop.

```
       DØ 10 LNUM=1, NET
       CALL BARK (ELK, ELQ)
       CALL ASEMBL (LNUM, 4, ELK, ELQ)
 10    CØNTINUE
  C    END ØF STAGE 3
```

Stage 4: Rotation Transformations

In the present problem, only the lower left node (Figure 12) requires rotation. Master degrees of freedom number 1 and 2 must be rotated by -45 degrees.

```
       CALL ROTATE(1, 1, 2, 0, -45., 0., 0.)
                            ↑         ↑   ↑
                            Quantities for 3-D problems are not used
  C    END ØF STAGE 4
```

Stage 5: Boundary Conditions

Displacements 2, 7, 8, 13, 14 are constrained, and the constraint vector starts in INTGR(1) (ICØN=1). Therefore:

```
      INTGR(1)=2
      INTGR(2)=7
      INTGR(3)=8
      INTGR(4)=13
      INTGR(5)=14
```

The constrained displacements are all prescribed to be zero. Only the nonzero prescribed forces need be input; these are 1,000 lb. each at the 15th and 17th degrees of freedom. Therefore:

```
C     PRESCRIBED DISPLACEMENTS
      REAL(IQ+1)=0.
      REAL(IQ+6)=0.
      REAL(IQ+7)=0.
      REAL(IQ+12)=0.
      REAL(IQ+13)=0.
C     ACCUMULATE PRESCRIBED FØRCES
      REAL(IQ+14)=REAL(IQ+14)+1000.
      REAL(IQ+16)=REAL(IQ+16)+1000.
      CALL BCØN
C     END ØF STAGE 5
```

Stage 6: Choleski Solution

```
      CALL FACTPD
      CALL SIMULQ(ENERGY)
C     END ØF STAGE 6
```

Stage 7: Inverse Rotation to Obtain Global Displacements in Global Cartesian Coordinates

A rotation of -45° was performed at node 1 before the boundary conditions were imposed. Since all element calculations are done with respect to the unrotated XY axis system, this rotation must be reversed before element stresses are calculated:

```
      CALL RØTATE(-1, 1, 2, 0, 45., 0., 0.)
C     END ØF STAGE 7
```

Stage 8: Calculation of Element Stresses

In this case, bar forces are to be calculated. Again, the user merely invokes the proper subroutines:

```
      DØ 20 LNUM=1, NET
      CALL XTRACT (LNUM, 4, ELQ)
      CALL BARF (LNUM, ELQ)
   20 CØNTINUE
C     END ØF PRØG
      STØP
      END
```

If the reader has grasped the material presented up to this point, he now has enough familiarity with FEABL to use it, albeit somewhat inefficiently. However, some additional degree of sophistication is desirable for the instructions which input problem data such as the assembly list. It is apparent that a straightforward set of instructions such as that given in the sample program is quite cumbersome, especially for problems involving more elements with more degrees of freedom per element. Improved programming techniques are discussed in the next section.

SECTION 5

ADVANCED TECHNIQUES WITH FEABL

5.1 Efficient Programming for Large Problems

When a structure is to be analyzed with a large number of elements and many degrees of freedom, there often occurs a definite trend toward regularity in the element set. The analyst should recognize two distinct forms of regularity, and he should be prepared to take advantage of each in writing his MAIN program.

First, consider the truss structure shown in Figure 13, with all bays having the same dimension vertically and horizontally.

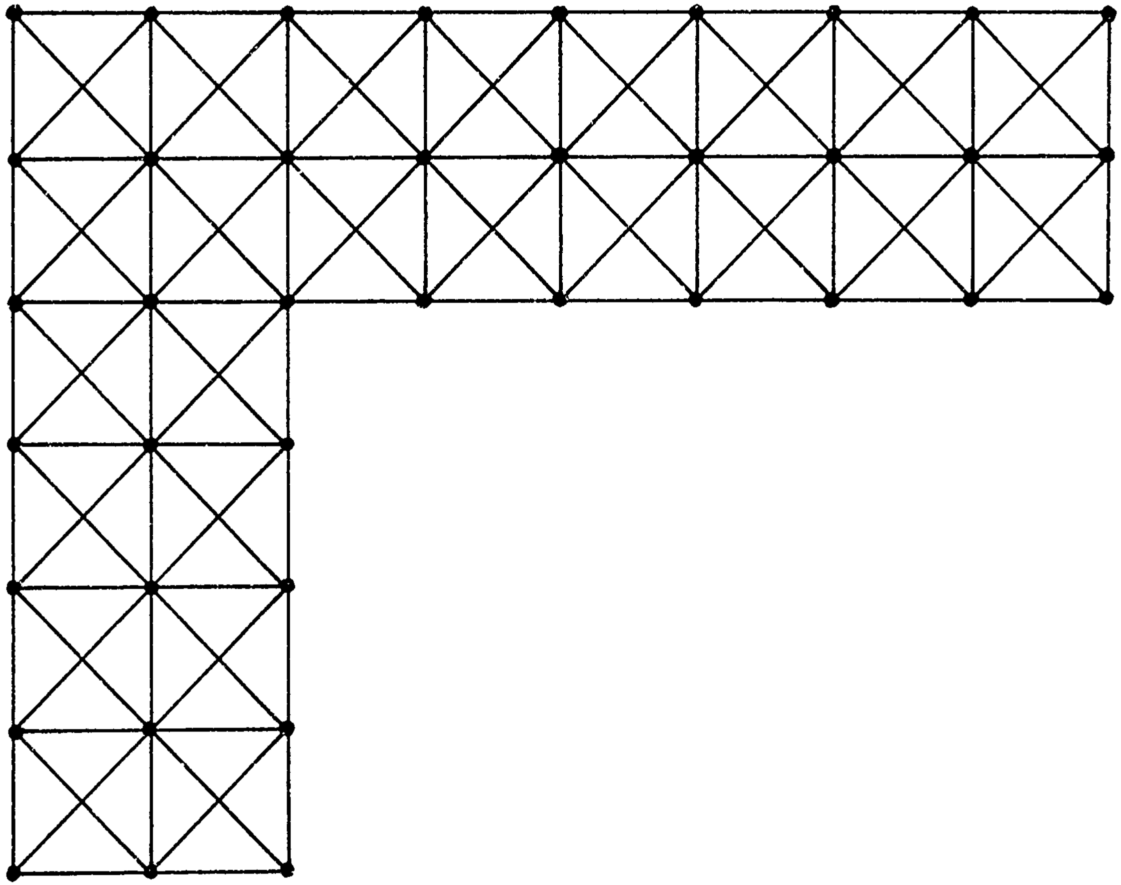


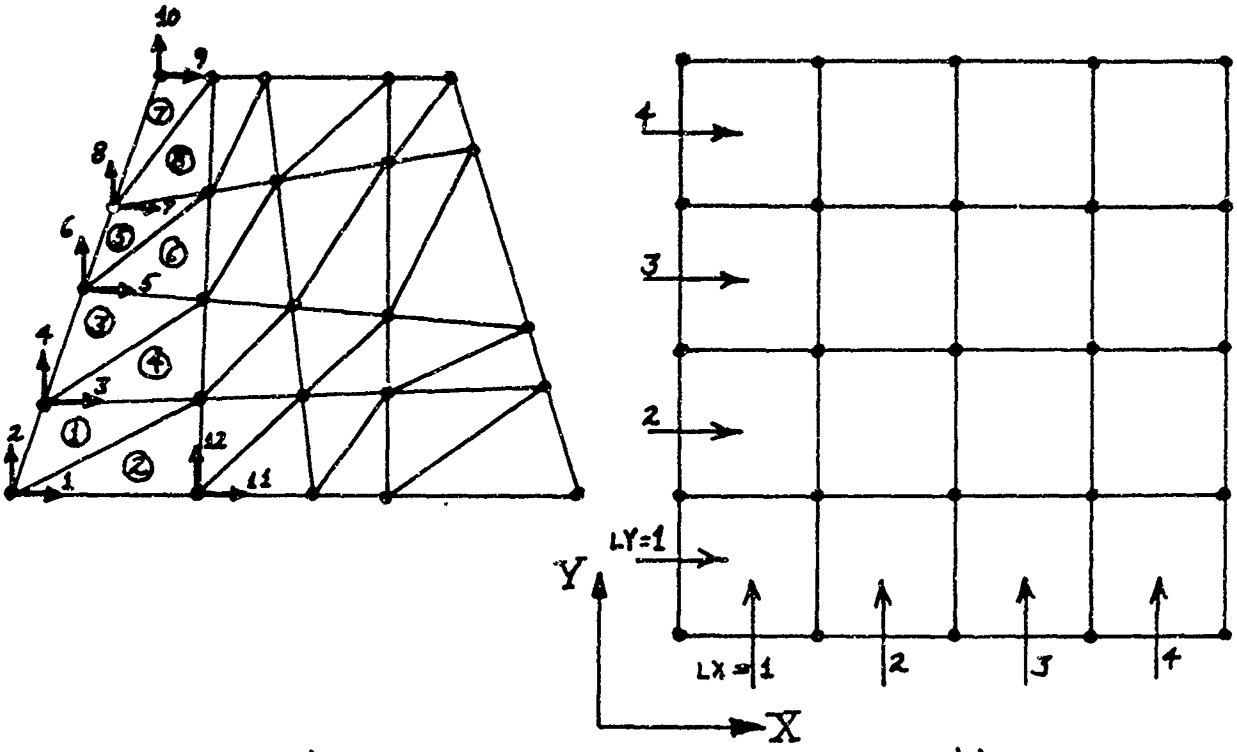
FIGURE 13

The bar elements may be said to possess a degree of regularity, in that there are only four different types of elements in the structure:

1. Horizontal bars
2. Vertical bars.
3. +45° diagonals.
4. -45° diagonals.

Since an element stiffness matrix depends upon nodal coordinates only through differences in the coordinate values (see beginning of Section 4), there will occur only four independent element stiffness matrices for the structure of Figure 13: one for each of the element types listed above. Thus, for example, the element stiffness matrix for a typical horizontal bar may be generated once and assembled repeatedly (32 times for the present case). The technique may be repeated again for the 30 vertical bars, 24 +45° diagonals and 24 -45° diagonals, achieving considerable savings in execution time.

The second form of regularity involves the way in which displacements and elements may be numbered. Often an element set is topologically equivalent to a rectangular or square array, even if it is not geometrically regular. For example, suppose the trapezoidal continuum shown in Figure 14(a) is to be analyzed in plane stress using 32 3-node triangle elements. For numbering purposes the element and displacement sets are topologically equivalent to the square net shown in Figure 14(b). It is then possible to generate the assembly list for the structure by a double DØ loop. Each element is "located" in the structure via an intersection of one of the element strings LX and one of the element strings LY. The element number and its master displacement numbers may be generated from the values of LX and LY. Taking advantage of numbering regularity enables the user to input large sections of the assembly list with relatively few FØRTRAN instructions.



(a) (b)

FIGURE 14

5.2 A Numbering Strategy for Planar Problems

Suppose the set of MxN rectangular plane stress elements shown in Figure 15 is to be assembled, where M, N may be quite large. If the numbering strategy shown in the figure is adopted, beginning with element 1 and displacements 1, 2 at the lower left corner and ending with element MN and displacements 2(M+1) (N+1)-1, 2(M+1) (N+1) at the upper right corner, then each element number can be given as a function of its LY, LX string coordinates:

LNUM=LY+N*(LX-1) where $1 \leq LY \leq N$ and $1 \leq LX \leq M$

In order to create similar functions for the element assembly list, a local numbering convention must be adopted. If the

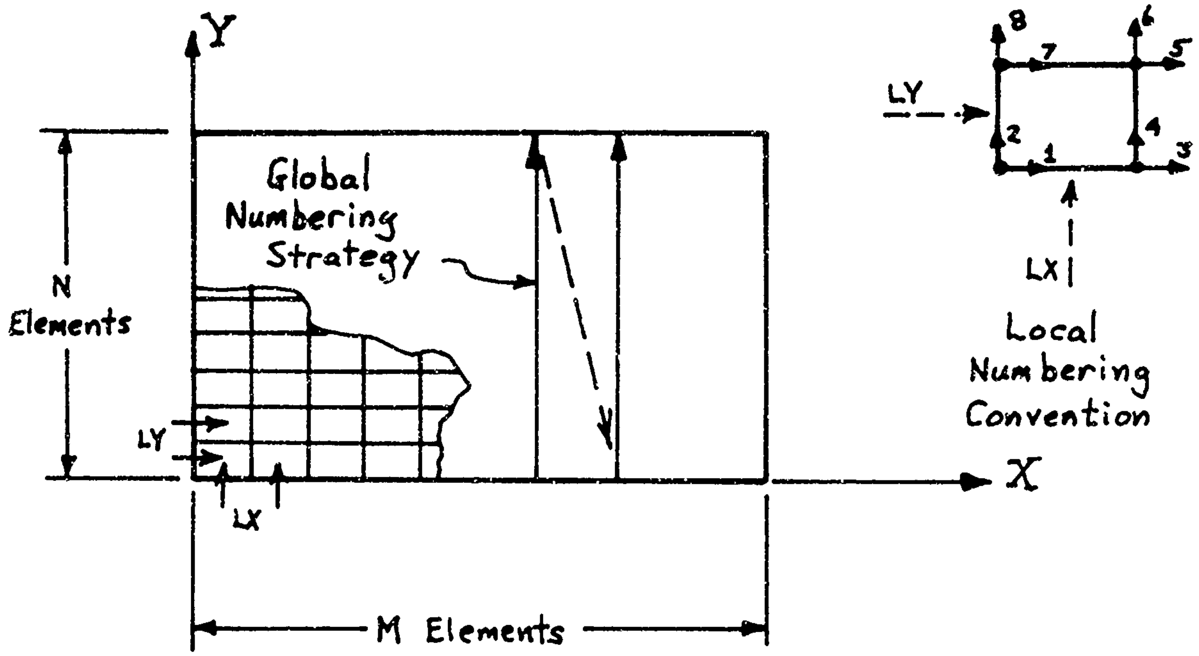


FIGURE 15

convention shown is Figure 15 is used, the global DOF numbers may be calculated as follows: Local displacements 1, 2 can be considered to lie on strings LX, LY; hence:

(Local #1)=2*LY-1+2*(N+1)*(LX-1)
(Local #2)=(Local #1)+1

Local displacements 7, 8 follow directly from the master numbering scheme:

(Local #7)=(Local #2)+1=(Local #1)+2
(Local #8)=(Local #7)+1=(Local #1)+3

Local displacements 3, 4 are shifted one LX string to the right; hence they are 2*(N+1) ahead of 1, 2:

(Local #3)=(Local #1)+2*(N+1)
(Local #4)=(Local #1)+2*(N+1)+1=(Local #2)+2*(N+1)

and finally:

(Local #5)=(Local #1)+2*(N+1)+2=(Local #7)+2*(N+1)
(Local #6)=(Local #1)+2*(N+1)+3=(Local #8)+2*(N+1)

With the algorithms derived above, it is quite easy to develop an efficient procedure to input the entire assembly list into the /DATA/ vector. The reader will recall that the pointer for element number LNUM is stored in INTGR(IMASTR+LNUM-1). One additional integer variable, called NEXT, is required. NEXT is to be incremented after completion of the input for one element, so that the next value of NEXT is the address of the next available location in the master assembly list, i.e., NEXT= the value of the pointer for the next element. The entire assembly list is then generated and input by the following set of FØRTRAN instructions:

```
        (Values of M and N defined elsewhere in program)
        .
        .
        .
    C   INITIALIZE PØINTER VALUE
        NEXT=IMASTR+NET
    C   LØØP ØVER ELEMENT STRINGS
        DØ 20 LX=1, M
        DØ 20 LY=1, N
        LNUM=LY+N*(LX-1)
        IPTR=IMASTR+LNUM-1
    C   ESTABLISH PØINTER FØR THE ELEMENT
        INTGR(IPTR)=NEXT
    C   CALCULATE JD1=1 LESS THAN MASTER NØ. ØF 1ST DØF, JD3=1
    C   LESS THAN MASTER NØ. ØF 3RD DØF
        JD1=2*(LY-1)+2*(N+1)*(LX-1)
        JD3=JD1+2*(N+1)
    C   ASSIGN DOF MASTER NOS. TO LOCATIONS NEXT, NEXT + 1,..., NEXT + 7
    C   IN /DATA/ VECTØR
        J=0
        K=0
        DØ 10 I=1,8
        INDEX=NEXT+I-1
        IF(I .GT. 2 .AND. I .LT. 7) GØ TØ 5
        J=J+1
        JD=JD1
        II=J
        GØ TØ 10
    5   JD=JD3
        K=K+1
        II=K
   10   INTGR(INDEX)=JD+II
    C   INCREMENT NEXT LØCATIØN
   20   NEXT=NEXT+8
    C   STØRE A ZERØ IN NEXT LØCATIØN (ASSUMING EXCESS STØRAGE
    C   IN MASTER ARRAY)
        IF (NEXT .LE. LMASTR) INTGR(NEXT)=0
        .
        .
        .
```

The last IF statement above is a good form of insurance for cases in which the user may have overestimated the number of words required for his master assembly list.

A similar scheme for structures consisting of triangle elements (Figure 14) can be developed if the "right-side-up" and "upside-down" elements are treated as separate sets.

The automatic generation technique derived above was developed for a structure with complete topological regularity; however, it may be extended to cover large portions of less regular structures with relatively little additional programming. There is also a hidden advantage: the numbering strategy adopted in Figure 15 is not only easy to produce, but also minimizes the population of the master stiffness matrix if none of the elements have mid-side nodes.

5.3 A Short Note on Automatic Data Generation

The reader will recall that the hypothetical user in Section 4 chose to read in his element coordinates and basic properties from data cards. This procedure can be time-consuming and expensive for problems involving large numbers of elements, especially when the element set geometry is not regular in the sense of Subsection 5.1 and Figure 13. However, it often happens that the element set geometry is regular in the sense that the element coordinates may be calculated from a general algorithm based upon the element string concept discussed in the previous subsection, i.e.:

```
DØ 20 LX=1, M
DØ 20 LY=1, N
LNUM=LY+N*(LX-1)
(X⃗=f⃗(LX, LY))
•
•
•
```

where $\vec{X}$={X1, Y1, X2, Y2,...} is the element coordinate vector and $\vec{f}$ is a floating point function of LX and LY. An algorithm of this type may be used to generate the data base, as it is needed, for calculation of element stiffness matrices and element stresses. This results in the trade-off of a slight increase in execution time and some decrease in required storage space, since it is not necessary to carry large vectors of global nodal coordinates in core.

5.4 A Handy Trick

There often occur plane elasticity problems in which a large number of degrees of freedom are to have prescribed displacements. For example, suppose a rectangular domain such as the one shown in Figure 15 is to be analyzed, and that all four edges of the domain are clamped. This means that 25 to 30 per cent to the total degrees of freedom will have prescribed displacements.

A significant amount of execution time may be saved by modifying the numbering scheme shown in Figure 15, so that the edge degrees of freedom have the largest global numbers. Let NDT be the total number of degrees and NFT be the total number of unconstrained degrees. Then the modified number scheme assigns:

1, 2,..., NFT

to the unconstrained degrees and:

NFT+1, NFT+2,..., NDT

to the degrees along the edges of the structure. Program stages 1 through 5 are completed in standard fashion.

However, just before factoring $\mathbf{K}$, the user may fool FEABL by inserting:

```
ITEMP=NDT
NDT=NFT
```

in his MAIN subprogram. Subroutines FACTPD/FACTSD and SIMULQ will then solve only for the unknown displacements 1, 2,..., NFT. At the beginning of Stage 7, when the full displacement vector may be required again, the user inserts:

```
NDT=ITEMP
```

The real value of this trick depends on a trade-off between execution time and core storage. The modified numbering scheme will result in a requirement for additional storage space for $\mathbf{K}$, over what is needed by the numbering scheme discussed in Subsection 5.2. The excess requirement is given by:

$$\Delta S \approx (NDT-NFT)\ (NDT-B)$$

where B is the average semi-bandwidth of the unconstrained part of $\mathbf{K}$.

5.5 Time-Saving Techniques for Design Studies

The "design study" approach to finite element analysis may take any of the following forms:

1. Consideration of a number of loading environments applied to a unique structure with unique displacement boundary conditions.
2. Analysis of a unique structure under various environments in which prescribed displacements are changed, as well as prescribed forces.
3. Consideration of variations in the structure itself to meet a given environment.

The first two categories are self-explanatory. An example of the third might be an analysis of a truss structure in which the compression bars are checked for buckling:

$$P_{Bar} < P_{cr} = \pi^2 EI/\ell^2$$

If any bar were found to be under a compressive load greater than its P_{cr}, the program might incorporate an algorithm for redesigning the bar and re-analyzing the new structure.

The above categories of problems can be studied using FEABL with only minor modifications. No changes are required in the data location interfaces and if a change to the process sequence interface is required, it involves only an intelligent application of the rules presented above. However, the analyst must be familiar with programming techniques required for communication between his computer's core and external storage devices (system disks, drums, tape units). These techniques vary from installation to installation and will not be discussed in specific terms here. The analyst should consult the operating manuals applicable to the system which he will be using.

Load environment case study is the easiest type of multi-solution problem to handle. Since the displacement boundary conditions are not varied, only one assembly and factoring of the master stiffness matrix need be done. The factored form of $\mathbf{K}$ is held in the /DATA/ vector while each prescribed vector is formed and the displacements are solved for. The quantity:

$$Q^{*} = \hat{Q}_{E} - K_{FD}\hat{u}_{D}$$

$\hat{Q}_{E}$= Assembled Element Equivalent Nodal Forces and the prescribed displacements $\hat{u}_{D}$ must be saved. These two quantities are found in the Force/Displacement Vector:

$$Q = \begin{Bmatrix} Q^{*} \\ \hat{u}_{D} \end{Bmatrix}$$

in the /DATA/ vector just after BCØN is called. (The prescribed external loads $\dot{Q}_{F}$, on the other hand, are not really needed until SIMULQ is called.) Figure 16 illustrates the modified FEABL process sequence which will accomplish this result.

The second category involves essentially the same techniques as the first. However, in this case the data which must be saved in external storage are:

1. $\hat{Q}_{E}$ - Assembled element equivalent nodal forces.
2. $\mathbf{K}$ - Assembled master stiffness matrix <u>prior to boundary condition application</u>.

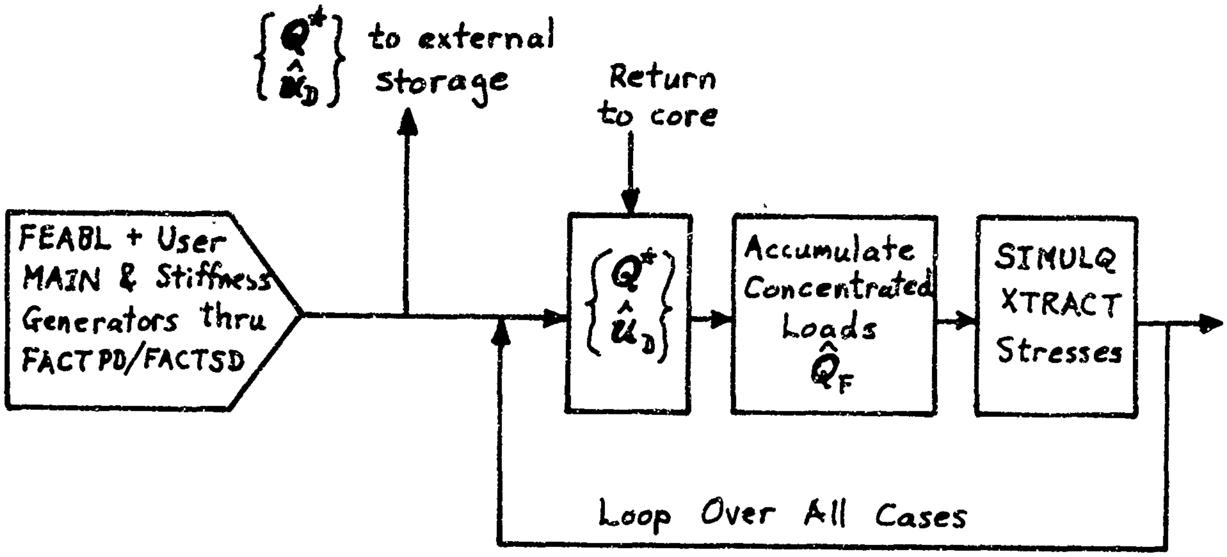

$\left\{\begin{matrix} Q^* \\ \hat{u}_D \end{matrix}\right\}$ to external storage
Return to core
FEABL + User MAIN & Stiffness Generators thru FACTPD/FACTSD
$\left\{\begin{matrix} Q^* \\ \hat{u}_D \end{matrix}\right\}$
Accumulate Concentrated Loads $\hat{Q}_F$
SIMULQ XTRACT Stresses
Loop Over All Cases

FIGURE 16

For each case the user must input the contents of the constraint vector, fetch $\hat{Q}_E$, accumulate $\hat{Q}_F$ and reset $\hat{u}_D$, fetch $\mathbf{K}$ and finally re-enter the standard FEABL process at the point where BCØN is called.

Significant time savings are gained in the third problem category if the design changes involve only a few elements in a structure composed of a large number of elements. Suppose that the displacement solution has been found from the force-displacement relations for the initial structure:

$$\mathbf{K}u = \hat{Q}_F + Q^*$$

During calculation of the element stresses, it is discovered that design changes are required in one or more elements; these changes will appear in the force-displacement relations as a modification $\Delta\mathbf{K}$ of the master stiffness matrix and (possibly) a modification $\Delta\hat{Q}_E$ of the assembled element nodal force vector.

The new force-displacement relations:

$$(\mathbf{K} + \Delta\mathbf{K})\, \boldsymbol{u}' = \hat{\boldsymbol{Q}}_F + \boldsymbol{Q}^* + \Delta\boldsymbol{Q}^*$$

must then be solved for the modified displacements $\boldsymbol{u}'$.

To program the design change technique with FEABL, the user must take the following nonstandard actions:

1. Output $\mathbf{K}$ and $\hat{\boldsymbol{Q}}_F + \hat{\boldsymbol{Q}}_E$ to external storage after assembly and input of prescribed quantities, but before calling BCØN.
2. After the initial displacement solution $\boldsymbol{u}$ has been obtained, transfer it to temporary core storage outside the /DATA/ vector. (The Force/Displacement Vector area in the /DATA/ vector will be required for accumulation of $\Delta\hat{\boldsymbol{Q}}_E$.) Zero the Force/Displacement Vector and the Master Stiffness Matrix Array.
3. Calculate stresses from $\boldsymbol{u}$, element-by-element. (The user will need his own version of XTRACT to extract the proper $\boldsymbol{u}_{el}$ from temporary core storage.) When a design change is required, calculate $\Delta\mathbf{k}_{el}$ and $\Delta\hat{\boldsymbol{Q}}_{el}$ and accumulate them using FEABL subroutine ASEMBL.
4. Apply rotation transformations (if any) to $\Delta\mathbf{K}$ and $\Delta\hat{\boldsymbol{Q}}_E$, reset entries of $\Delta\hat{\boldsymbol{Q}}_E$ to zero where displacements are prescribed.
5. Fetch $\mathbf{K}$ and $\hat{\boldsymbol{Q}}_F + \hat{\boldsymbol{Q}}_E$ from external storage and accumulate them to $\Delta\mathbf{K}$ and $\Delta\hat{\boldsymbol{Q}}_E$. Apply boundary conditions and solve for $\boldsymbol{u}'$.

The above process may be repeated in an iterative design process, with $\mathbf{K} + \Delta\mathbf{K}$, $\hat{\boldsymbol{Q}}_F + \boldsymbol{Q}^* + \Delta\boldsymbol{Q}^*$ at the beginning of each new design step playing the role of the initial values $\mathbf{K}$, $\hat{\boldsymbol{Q}}_F + \boldsymbol{Q}^*$. This procedure is referred to as iterative updating, and is also applicable to nonlinear elastic and elastic-plastic analysis of continua. In these types of analysis the "design change" results from following a material stress-strain curve and/or testing the satisfaction of a yield conditon (e.g., the Mises-Hencky criterion).

5.6 Substructuring with FEABL

The FEABL software system has been designed primarily for in-core solutions. Up to 1,500 degrees of freedom can be handled on currently available hardware with 500 KBYTE (125 KWord) memory. FEABL's in-core capability may be extended by means of the substructuring technique outlined briefly here.

Figure 17 illustrates a domain which has been divided into a small number of substructures. The desired stress solution accuracy is on a scale much smaller than the substructure dimension,

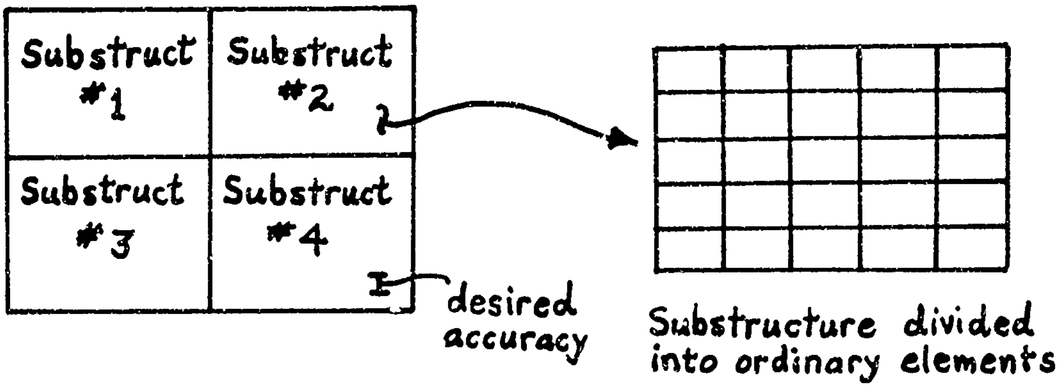


FIGURE 17

so each substructure is subdivided further into ordinary elements. Let the subscripts I and B refer, respectively, to the interior and boundary degrees of freedom in a substructure. Then the force-displacement relations for the substructure may be partitioned into:

$$\begin{bmatrix} \mathbf{K}_{BB} & \mathbf{K}_{BI} \\ \mathbf{K}^{T}_{BI} & \mathbf{K}_{II} \end{bmatrix} \begin{Bmatrix} \boldsymbol{u}_B \\ \boldsymbol{u}_I \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{Q}_B \\ \boldsymbol{Q}_I \end{Bmatrix}$$

The interior degrees of freedom may be eliminated by the process of static condensation, which transforms the substructure force-displacement relations to the form:

$$\mathbf{K}^{c}_{BB}\, \boldsymbol{u}_B = \left(\mathbf{K}_{BB} - \mathbf{K}_{BI} \mathbf{K}^{-1}_{II} \mathbf{K}^{T}_{BI} \right) \boldsymbol{u}_B = \boldsymbol{Q}_B - \mathbf{K}_{BI} \mathbf{K}^{-1}_{II} \boldsymbol{Q}_I = \boldsymbol{Q}^{c}_{B}$$

Each substructure may now be considered as a "superelement" having degrees of freedom only on the interelement boundaries, as shown in Figure 18. The quantities $\mathbf{K}^{c}_{BB}$ and $\boldsymbol{Q}^{c}_{B}$ are then assembled in the same manner as the stiffness matrices and equivalent nodal force vectors for ordinary elements. After the displacement solution on the substructure boundaries has been obtained, the original force-displacement relations for each substructure may be used, with $\boldsymbol{u}_B$ as prescribed displacements, to obtain the interior solution.

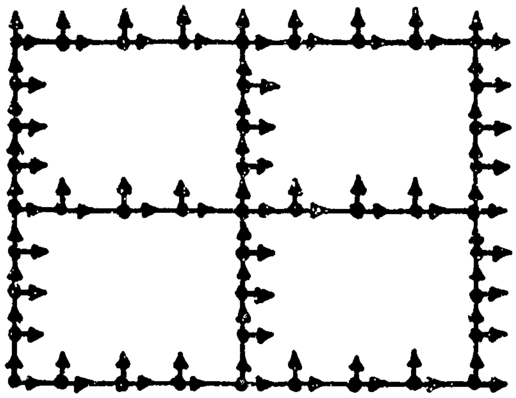


FIGURE 18

In addition to extension of FEABL's problem size capability, the substructuring technique may also be used to reduce roundoff error and to save execution time. All of these improvements depend upon optimization of the relative numbers of substructures and of ordinary elements within the substructures. Use of the substructuring technique with FEABL software will be discussed in detail in a future publication.

5.7 Error Estimation Methods

Rigorous mathematical proofs exist showing that a finite element analysis for the stress and displacement distributions in a structure converges to the exact solution as the number of elements in a given domain is increased, provided only that certain easily satisfied restrictions are obeyed (Ref.3). However, since all calculations done in a digital computer are imprecise, errors due to roundoff will occur. It has been shown (Ref. 4) that roundoff error increases in an extremely complex way as the number of elements is increased. Therefore, it is advisable to resort to some numerical method which will produce a reasonable estimate of the error, and which does not depend upon the details of what types of elements are used or what boundary conditions are applied to the structure. Four methods are presented here.

5.7.1 Irons' Energy Variance Criterion

The energy criterion proposed by Irons (Ref. 5) is the most economical, requiring no more time than the calculation of the rounding error parameter discussed in Subsection 3.2.6. The diagonal entries K_{ii} of the master stiffness matrix must be saved, either in temporary core storage or on an external unit. After the displacement solution has been obtained, the energy variance can be calculated from:

$$\mathcal{E} = 2^{-p+1}\sqrt{B}\,\frac{\sum_{i=1}^{NDT} K_{ii}\,(u_i)^2}{\sum_{i=1}^{NDT}\sum_{j=1}^{NDT} K_{ij}\,u_i u_j}$$

where p is the computer precision in decimal places and B is the average semi-bandwidth of $\mathbf{K}$. The value of B can be obtained easily from the address index parameters*:

B=FLØAT(LK+1-IK)/FLØAT(NDT)

One half the value of the denominator of the energy variance expression is provided to the user by FEABL subroutine SIMULQ in the argument ENERGY of that subroutine.

Some care must be exercised in applying Irons' energy criterion. The user will obtain unrealistically large values of $\mathcal{E}$ in problems in which the structure is constrained very lightly, and in which large amounts of "self-energy" can be stored when a single degree of freedom is displaced. The cantilever beam is a good example of a structure to which the Irons' criterion cannot be applied.

5.7.2 The Residual Force Method

Calculation of residual forces provides a more detailed picture of the distribution of errors through the structure. The master stiffness matrix may be saved in external storage for this purpose immediately after calling BCØN. Approximate values of the reaction forces at degrees where displacements were prescribed may be obtained as well by saving $\mathbf{K}$ just prior to calling BCØN.

[*] FLØAT is an IBM library function which converts integers to floating point numbers.

Let u^{*}_{F} be the approximate displacement solution obtained from the constrained force-displacement relations:

$$\begin{bmatrix} \mathbf{K}_{FF} & 0 \\ 0 & \mathbf{I} \end{bmatrix} \begin{Bmatrix} u_F \\ \hat{u}_D \end{Bmatrix} = \begin{Bmatrix} \hat{Q}_F - \mathbf{K}_{FD}\hat{u}_D \\ \hat{u}_D \end{Bmatrix}$$

Then:

$$Q^{*} = \begin{Bmatrix} Q^{*}_{F} \\ Q^{*}_{D} \end{Bmatrix} = \mathbf{K} \begin{Bmatrix} u^{*}_{F} \\ \hat{u}_D \end{Bmatrix}$$

is the approximate force vector. When $\mathbf{K}$ has been returned to core after the displacement solution has been obtained, Q^{*} can be calculated by the following algorithm:

```
      DIMENSIØN R(500)
C     VECTØR R MUST ALLØW ENØUGH STØRAGE FØR THE FULL
C     FØRCE VECTØR Q-STAR IF REACTIØN FØRCES ARE
C     DESIRED.  ØTHERWISE, ENØUGH FØR THE UNCØNSTRAINED
C     DEGREES ØF FREEDØM MUST BE ALLØWED
C     ALLOCATE EXTERNAL FILES
      .
      .
      .
      (Standard FEABL process sequence, except where noted)
      .
      .
      .
      .
C     K MATRIX TØ EXTERNAL STØRAGE
      WRITE (...) (REAL(I), I=IK,LK)
C     FØRCE/DISPL MUST ALSØ BE SAVED FØR CØMPARISØN LATER
      WRITE (...) (REAL(I), I=IQ, LQ)
      CALL BCØN
      CALL FACTPD
      CALL SIMULQ(STRE)
      .
      .
      .
      (Stress solution, if desired)
      .
      .
      .
C     RETURN K MATRIX FØR CALCULATIØN ØF Q-STAR
      READ (...) (REAL(I), I=IK, LK)
C     INITIALIZE CØNSTRAINED RØW PØINTER AT 1ST NØNZERØ RØW
      DØ 50 II=ICØN, LCØN
      IF(INTGR(II) .EQ. 0) GØ TØ 50
      NC=II
      GØ TØ 60
   50 CØNTINUE
```

```
C   INITIALIZE R VECTØR PØINTER
    60 NR=1
C   LØØP ØVER DEGREES ØF FREEDØM, SKIPPING CØNSTRAINED DEGREES
    DØ 70 IRØW=1, NDT
    IF(INTGR(NC) .NE. IRØW) GØ TØ 71
C   UPDATE RØW PØINTER
    NC=NC+1
    GØ TØ 70
C   FØRM SUMMATIØN ØF K(IRØW, J)* U(J), J=LNZ TØ NDT
 71 INIT=ILNZ+IRØW-1
    INIT=INTGR(INIT)
    KK=IKØUNT+IRØW-1
    KK=INTGR(KK)
    SUM=0.
    DØ 72 J=INIT, IRØW
    KADR=KK+J
    JJ=IQ+J-1
 72 SUM=SUM+REAL(KADR)* REAL(JJ)
C   REMAINDER ØF SUM MUST TAKE K ENTRIES FRØM CØL IRØW
    INIT=IROW+1
    IF(INIT .GT. NDT) GØ TØ 74
    DØ 73 J=INIT, NDT
C   MAKE SURE RØW J HAS AN ENTRY IN CØL IRØW
    JJ=ILNZ+J-1
    IF(INTGR(JJ) .GT. IRØW) GØ TØ 73
    KADR=IKØUNT+J-1
    KADR=INTGR(KADR)+IRØW
    JJ=IQ+J-1
    SUM=SUM+REAL(KADR)*REAL(JJ)
 73 CØNTINUE
C   PLACE SUM IN NEXT AVAIL R LØCATIØN AND UPDATE R PØINTER
 74 R(NR)=SUM
    NR=NR+1
 70 CØNTINUE
C   RETURN ØRIGINAL Q TØ CØRE FØR CØMPARISØN
    READ (...) (REAL(I), I=IQ, LQ)
C   CALCULATE RESIDUAL FØRCES Q-(Q-STAR)=Q-R
C   REINITIALIZE NC AND NR
    NC=II
    NR=1
    DØ 80 IRØW=1, NDT
    IF(INTGR(NC) .NE. IRØW) GØ TØ 81
    NC=NC+1
    GØ TØ 80
 81 JJ=IQ+IRØW-1
    R(NR)=REAL(JJ)-R(NR)
    WRITE (KW, 500) IRØW, R(NR)
500 FØRMAT (20H RESIDUAL FØRCE NØ.,I6, 1X, 1H=, E10.3)
    NR=NR+1
 80 CØNTINUE
```

In the above algorithm, the vector of residual forces:

$$\mathbf{R}_F = \hat{Q}_F - Q_F^* = \Delta Q_F$$

has been calculated. The entries of $\mathbf{R}_F$ provide the detailed picture mentioned at the beginning of this subsection. Another useful parameter for overall error measurement is the force vector magnitude ratio:

$$r_F = \left[\frac{\Sigma(R_i^2)}{\Sigma(\hat{Q}_i - \Sigma K_{ij} u_j)^2}\right]^{1/2} = \frac{|\Delta Q_F|}{|\hat{Q}_F|}$$

where the summations extend only over the unconstrained degrees of freedom.

5.7.3 Re-Solution for Residual Displacements

Although the residual force vector is fairly easy to calculate, the interpretation of its meaning is not a trivial task. The displacement solution must certainly be judged acceptable if, for example, there are many residual forces on the order of 1 lb. at degrees where forces of 1,000 lb. were applied originally. However, a common situation in finite element analysis is that the applied force is zero at many degrees of freedom. What does a 1 lb. residual force mean at these points? The averaged measure r_F presented in the previous section relieves this detailed interpretation problem to some degree. However, the averaged measure of greatest interest to the analyst is the displacement vector magnitude ratio:

$$d_F = \frac{|\Delta u_F|}{|u_F|}$$

Unfortunately, no simple relation exists between d_F and the force vector magnitude ratio r_F. If $\mathbf{K}$ is an ill-conditioned matrix, d_F may in fact be much larger than r_F.

If the analyst is willing to spend some additional computing time, the displacement residuals and an approximate calculation of d_F may be obtained by re-solution. This technique requires an additional external storage file capable of holding $\mathbf{K}$. Just before $\mathbf{K}$ is returned to core in the algorithm of Subsection 5.7.2, its factored form $\mathbf{LDL}^T$ is read into this extra file. Then, picking up where the previous algorithm ended, the original displacement vector magnitude is calculated, $\mathbf{LDL}^T$ is returned to core and the contents of $\mathbf{R}$ are transferred to the proper locations in the force/displacement block of the /DATA/ vector. Re-solution is now done simply by calling SIMULQ again, after which $\Delta \boldsymbol{u}_F$ will be found in the force/displacement block. The displacement vector magnitude ratio obtained from this procedure is actually:

$$d_F^* = \frac{|\Delta u_F|}{|u_F^*|}$$

and $d_F^* \neq d_F$ unless $|\Delta \boldsymbol{u}_F| << |\boldsymbol{u}_F^*|$.

5.7.4 The Method of Rigid Body Modes

A somewhat less cumbersome technique for error measurement can be employed when a structure is modeled by elements which contain a full set of rigid body modes in their assumed displacement fields. Let

$$\boldsymbol{r} = \begin{Bmatrix} \boldsymbol{r}_F \\ \hat{\boldsymbol{r}}_D \end{Bmatrix}$$

be any rigid body displacement vector for the whole structure (e.g., unit vertical translation at every node). Then if $\boldsymbol{r}$ is introduced into the unconstrained force-displacement relations and the right hand side is calculated, there will result:

$$\mathbf{K}\boldsymbol{r} = \begin{Bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{Bmatrix}$$

to the accuracy of the user's computer. Conversely, solution of the constrained force-displacement relations:

$$\begin{bmatrix} \mathbf{K}_{FF} & 0 \\ 0 & \mathbf{I} \end{bmatrix} \begin{Bmatrix} \boldsymbol{u}_F \\ \hat{\boldsymbol{r}}_D \end{Bmatrix} = \begin{Bmatrix} -\mathbf{K}_{FD}\hat{\boldsymbol{r}}_D \\ \hat{\boldsymbol{r}}_D \end{Bmatrix}$$

would result in $\boldsymbol{u}_F = \boldsymbol{r}_F$ to the accuracy of the computer if there were no roundoff error. Now, since the exact $\boldsymbol{r}_F$ can be inferred from $\hat{\boldsymbol{r}}_D$ merely by inspection, the above problem may be solved as an auxiliary to the real problem, and the error measure:

$$d_{RB} = \frac{|\boldsymbol{r}_F - \boldsymbol{u}_F|}{|\boldsymbol{r}_F|}$$

may be calculated. The rigid body mode technique requires the saving only of $\mathbf{K}$ (before constraint). A simple algorithm will serve to calculate $\boldsymbol{r}_F$ and $\boldsymbol{r}_F - \boldsymbol{u}_F$, and the additional execution time required amounts only to calling each of BCØN, FACTPD (or FACTSD) and SIMULQ once extra. Also, the lengthy residual force and re-solution algorithm in the MAIN program is avoided. Recent tests of the method of rigid body modes on a cantilever beam have shown that when $d_F<0.1$, $d_{RB} << d_F$. However, when $d_F>0.1$ (the region of primary interest) d_{RB} performs as well as d_F^*.

REFERENCES

1. Orringer, O. "The Effect of Cross Section Stress Concentration on the Compressive Strength of a Unidirectional Fiber Composite." ASRL TR 162-4 (in preparation)

2. Mack, E. W., Berg, B. A., and Witmer, E. A. "An Improved Discrete-Element Analysis and Program for the Linear-Elastic Static Analysis of Meridionally-Curved, Variable-Thickness, Branched Thin Shells of Revolution Subjected to General External Mechanical and Thermal Loads, Part 2-The SABOR 4 Program." Massachusetts Institute of Technology, Aeroelastic and Structures Research Laboratory, ASRL TR 146-4, Part 2 (also SAMSO TR 68-310, Part 2), March 1968. (AD 840 614L).

3. Tong, P. and Pian, T. H. H. "The Convengence of Finite Element Method in Solving Linear Elastic Problems." Int'l. J. Solids Structures, Vol. 3, 1967. pp. 865-879.

4. Tong, P. "On the Numerical Problems of the Finite Element Methods." Study No. 5, Symposium on Computer-Aided Engineering, University of Waterloo, Ontario, Canada, 1970.

5. Irons, B. M. and Kan, D. K. Y. "Equation-Solving Algorithms for the Finite-Element Methods." Paper No. V-4-2, University of Illinois Conference on Numerical Methods in Structural Analysis, 1971.

APPENDIX A

CØMMØN AREAS DATA REQUIREMENTS

The following table summarizes what information is expected by each FEABL subroutine in the four control parameter CØMMØN areas and in the /DATA/ vector CØMMØN area.

SUBROUTINE NAME CØMMØN AREA	ASEMBL	BCØN	FACTPD/FACTSD	ØRK	RØTATE	SETUP	SIMULQ	XTRACT
/IO/ (Printer Code KW Only)	✓	✓	✓	✓	✓	✓	✓	
/SIZE/ NET, NDT		✓	✓	✓	✓	✓	✓	
/BEGIN/ Address index parameters	✓	✓	✓	✓	✓		✓	✓
/END/ Address index parameters		✓	✓	✓ Except LK			✓	
/DATA/ Vector:								
1. Constraint Vector		✓	✓			(c)	✓	
2. Address Count Vector	✓	✓	✓		✓		✓	
3. LNZ Vector	✓	✓	✓		✓		✓	
4. Assembly List	✓			✓				✓
5. F/D Vector	(a)	✓			Only $\hat{Q}_E$ (b)		✓	$\mathcal{U}$
6. **K** Matrix	(a)	✓	✓		✓		LDL^T	

(a) These blocks are zeroed by ØRK before the first element is assembled.

(b) $\hat{\boldsymbol{Q}}_E$ = Vector of assembled element equivalent nodal forces.

(c) Block zeroed by SETUP

APPENDIX B

APPLYING SHOEHORN AND STOPWATCH

This appendix contains data from which the user may estimate the total amount of core storage required by a FEABL-based program and the approximate execution time (CPU time) the run will take. Such estimates will prove useful aids in making trade-off decisions. The data given in this appendix is based on runs done on an IBM 370/155 using the FØRTPAN G compiler. The numbers will vary somewhat from one machine or compiler to another.

B.1 Estimation of Core Storage Requirement

The following table gives the length of each FEABL subroutine in BYTES (as compiled in FØRTRAN G) and words, and the deck size. On IBM 360 and 370 series hardware, core storage calculations are normally done in terms of BYTES, while words are used on many other hardware systems. The total storage requirement for programs plus data can be estimated as follows:

Subroutine Name	Words	BYTES	No. of Cards in Deck*
ASEMBL	410	1,642	58
BCØN	599	2,396	125
FACTPD/FACTSD	669	2,674	126
ØRK	517	2,070	106
RØTATE	1,858	7,430	316
SETUP	726	2,904	94
SIMULQ	600	2,398	127
XTRACT	130	522	23
FEABL Software-Total	5,509	22,036	975

*Includes all comment cards

Item:	KWords	KBYTES
1. FEABL Software	5.5	22.1
2. User programs (a)	2.5	10.0
3. System library subprograms (b)	5.1	22.0
4. /DATA/ Vector (c)	(L)	(4L)
5. Four control parameter /CØMMØN/ areas	--	0.1
Totals	13.1+(L)	53.2+(4L)

(a) Estimate for a typical analysis with a MAIN program and two generator subroutines.

(b) Includes library functions such as SIN, CØS, SQRT and systems management subroutines.

(c) L=the dimension of the /DATA/ vector in KWords (1000 words).

B.2 Estimation of CPU Time Requirement

The process of factoring the master stiffness matrix into its triple product:

$$\mathbf{K} = \mathbf{L}\mathbf{D}\mathbf{L}^T$$

is the primary time consumer in any finite element analysis. Some study of the algorithms in FEACL subroutine FACTPD/FACTSD will convince the reader that the CPU time consumed is proportional to NB^2, where N is the total number of unconstrained DOF in the assembled structure and B is the average semi-bandwidth of the master stiffness matrix. B may be calculated approximately from N and the population density of $\mathbf{K}$:

$$B \doteq \frac{(N+1)P}{2}$$

where

$$P = \text{Population Density} = \frac{\text{Total No. of Stored Entries}}{\text{Total Entries in Full Lower Triangle}}$$

Experience on the IBM 370/155 indicates that the required CPU time is given approximately by:

$$\text{time} \doteq NB^2 x10^{-6} \text{ minutes}$$

APPENDIX C

FØRTRAN IV LISTING OF FEABL SOFTWARE

The eight subroutines of the FEABL software system are listed in alphabetical order in this appendix:

The code is for FEABL Version 1 Release 1, with a 10,000-word /DATA/ vector. The following actions will convert the program to Version 2 ("On-Line") as explained in Section 2.3.

SUBROUTINE	Change DIMENSIØNs of REAL, INTGR to 2	Delete Declaration
ASEMBL	Card No. 0007	Card Nos. 0017 and 0020
BCØN	0007	0012 and 0015
FACTPD/FACTSD	0008	0013 and 0016
ØRK	0015	0020 and 0023
RØTATE	0007	0013 and 0017
SETUP	0007	0013 and 0015
SIMULQ	0007	0012 and 0015
XTRACT	0007	0011 and 0013

```
      SUBROUTINE ASEMBL(LNUM,NDE,ELK,ELQ)                               ASEM0001
C***********************************************************************ASEM0002
C FINITE ELEMENT ANALYSIS BASIC LIBRARY SUBRCUTINE                      ASEM0003
C AEROELASTIC AND STRUCTURES RESEARCH LABCRATORY                        ASEM0004
C MASSACHUSETTS INSTITUTE OF TECHNOLOGY                                 ASEM0005
C***********************************************************************ASEM0006
      DIMENSION REAL(10000),INTGR(10000)                                ASEM0007
      DIMENSION ELK(NDE,NDE), ELQ(NDE)                                  ASEM0008
C******************************                                         ASEM0009
C CHANGE THIS DIMENSION STATEMENT IF YOU HAVE RECEIVED AN ERROR MESSAGE ASEM0010
C INDICATING LENGTH OF MNUM VECTOR HAS BEEN EXCEEDED.  ALSO CHANGE      ASEM0011
C INSTRUCTION INDICATED BELOW                                           ASEM0012
      DIMENSION MNUM(100)                                               ASEM0013
C******************************                                         ASEM0014
      COMMON /IO/ KR, KW, KP, KT1, KT2, KT3                             ASEM0015
      COMMON /BEGIN/ ICON,IKOUNT,ILNZ,IMASTR,IQ,IK                      ASEM0016
      COMMON /DATA/ REAL                                                ASEM0017
C VERSION 1 RELEASE 1 AUGUST 1972                                       ASEM0018
C                                                                       ASEM0019
      EQUIVALENCE (REAL(1), INTGR(1))                                   ASEM0020
C PRINT CONTROL                                                         ASEM0021
  901 FORMAT(1H0,I10,27H DOF/ELEMENT EXCEEDS LENGTH,I4,26H SPECIFIED FORASEM0022
     1 MNUM VECTOR,/,1X,44HERROR OCCURED CURING ASSEMBLY OF ELEMENT NO.,ASEM0023
     2 I10,/,1X,93HUSER MUST CORRECT CALL ARGUMENTS IN MAIN OR CHANGE LEASEM0024
     3NGTH OF MNUM VECTOR IN SUBROUTINE ASEMBL,/,1X,20HEXECUTION TERMINAASEM0025
     4TED)                                                              ASEM0026
      IKOUM1 = IKOUNT-1                                                 ASEM0027
      IQM1 = IQ-1                                                       ASEM0028
C******************************                                         ASEM0029
C VALUE OF LENGTH MUST = DIMENSION OF MNUM                              ASEM0030
      LENGTH = 100                                                      ASEM0031
C******************************                                         ASEM0032
      IF (NDE .LE. LENGTH) GO TO 1                                      ASEM0033
      WRITE (KW,901) NDE, LENGTH, LNUM                                  ASEM0034
      STOP                                                              ASEM0035
C GET MASTER ROW/COL NOS FOR ELEMENT AND STORE IN MNUM                  ASEM0036
```

```
    1 INDEX = IMASTR+LNUM-1                                              ASEM0037
      DO 2 I = 1,NDE                                                     ASEM0038
      J = INTGR(INDEX)+I-1                                               ASEM0039
    2 MNUM(I) = INTGR(J)                                                 ASEM0040
C LOOP OVER ROWS OF ELQ AND OVER LOWER TRIANGLE OF ELEMENT K MATRIX       ASEM0041
      DO 4 LROW = 1,NDE                                                  ASEM0042
      INDEX = IQM1+MNUM(LROW)                                            ASEM0043
C ASSEMBLE ELEMENT EQUIVALENT NODAL FORCE INTO Q VECTOR                  ASEM0044
      REAL(INDEX) = REAL(INDEX)+ELQ(LROW)                                ASEM0045
      DO 4 LCOL = 1,LROW                                                 ASEM0046
      MROW = MNUM(LROW)                                                  ASEM0047
      MCOL = MNUM(LCOL)                                                  ASEM0048
      IF (MROW .GE. MCOL) GO TO 3                                        ASEM0049
      MROW = MNUM(LCOL)                                                  ASEM0050
      MCOL = MNUM(LROW)                                                  ASEM0051
C CALCULATE ABSOLUTE ADDRESS OF K(MROW,MCOL)                             ASEM0052
    3 INDEX = IKQUM1+MROW                                                ASEM0053
      KADR = INTGR(INDEX)+MCOL                                           ASEM0054
C ASSEMBLE STIFFNESS COEFFICIENT                                         ASEM0055
    4 REAL(KADR) = REAL(KADR)+ELK(LROW,LCOL)                             ASEM0056
      RETURN                                                             ASEM0057
      END                                                                ASEM0058
      SUBROUTINE BCON                                                   BCON0001
C***********************************************************************BCON0002
C FINITE ELEMENT ANALYSIS BASIC LIBRARY SUBROUTINE                      BCON0003
C AEROELASTIC AND STRUCTURES RESEARCH LABORATORY                        BCON0004
C MASSACHUSETTS INSTITUTE OF TECHNOLOGY                                 BCON0005
C***********************************************************************BCON0006
      DIMENSION REAL(10000),INTGR(10000)                                BCON0007
      COMMON /IO/ KR, KW, KF, KT1, KT2, KT3                             BCON0008
      COMMON /SIZE/ NET, NDT                                            BCON0009
      COMMON /BEGIN/ ICON,IKOUNT,ILNZ,IMASTR,IQ,IK                      BCON0010
      COMMON /END/ LCON,LKOUNT,LLNZ,LMASTR,LQ,LK                        BCON0011
      COMMON /DATA/ REAL                                                BCON0012
C VERSION 1 RELEASE 1 AUGUST 1972                                       BCON0013
C                                                                       BCON0014
      EQUIVALENCE (REAL(1), INTGR(1))                                   BCON0015
C PRINT CONTROL                                                         BCON0016
  901 FORMAT(79H0DISPLACEMENT CONSTRAINTS HAVE BEEN APPLIED TO THE FOLLOBCON0017
     1WING DEGREES OF FREEDOM,/,5X,7HDOF NO.,2X,12HDISPLACEMENT)        BCON0018
  902 FORMAT (2X,I10,2X,E10.3)                                          BCON0019
  903 FORMAT(1X,72H**********************ABOVE DOF NUMBER APPEARS TWICEBCON0020
     1 IN CONSTRAINT LIST,/,1X,85HEXECUTION TERMINATED IN SUBROUTINE BCOBCON0021
     2N DUE TO POSSIBLITY OF BOUNDARY CONDITION ERROR)                  BCON0022
  904 FORMAT(62H0YOUR STRUCTURE IS FLYING FREE. PLEASE CONSTRAIN IT NEXTBCON0023
     1 TIME.,/,1X,28HEXECUTION TERMINATED IN BCON)                      BCON0024
      IKOUM1 = IKOUNT-1                                                 BCON0025
      ILNZM1 = ILNZ-1                                                   BCON0026
      IQM1 = IQ-1                                                       BCON0027
C PRINT ENTRY MESSAGE                                                   BCON0028
      WRITE (KW,901)                                                    BCON0029
C ORDER THE CONSTRAINT ROW NUMBERS IN ASCENDING SEQUENCE                BCON0030
      LAST = LCON-1                                                     BCON0031
       IF (LCON .GT. 0) GO TO 1                                         BCON0032
       WRITE (KW,904)                                                   BCON0033
        STOP                                                            BCON0034
    1 IFLAG = 0                                                         BCON0035
      DO 2 I = ICON,LAST                                                BCON0036
      IF (INTGR(I+1) .GE. INTGR(I)) GO TO 2                            BCON0037
      J = INTGR(I)                                                     BCON0038
      INTGR(I) = INTGR(I+1)                                            BCON0039
      INTGR(I+1) = J                                                   BCON0040
      IFLAG = 1                                                        BCON0041
    2 CONTINUE                                                         BCCN0042
      IF (IFLAG .EQ. 1) GO TO 1                                        BCON0043
C CHECK TO SEE IF ANY ROW NUMBERS HAVE BEEN ENTERED IN CONSTRAINT      BCON0044
C VECTOR - ABORT THE RUN IF NONE HAVE BEEN                             BCON0045
      J = 0                                                            BCON0046
      DO 100 I = ICON,LCON                                             BCON0047
  100 J = J+INTGR(I)                                                   BCCN0048
      IF (J .GT. 0) GO TO 200                                          BCON0049
      WRITE (KW,904)                                                   BCON0050
      STOP                                                             BCCN0051
C OUTPUT CONSTRAINT LIST                                               BCON0052
  200 DO 4 I = ICON,LCON                                               BCCN0053
      IF (INTGR(I) .EQ. 0) GO TO 4                                     BCON0054
C CHECK FOR REPEATED DOF AFTER 1ST CNE                                 BCON0055
      IF (I .EQ. ICON) GO TO 3                                         BCON0056
      IF (INTGR(I) .NE. INTGR(I-1)) GO TO 3                            BCON0057
      WRITE (KW,903)                                                   BCON0058
      STOP                                                             BCON0059
    3 J = IQM1+INTGR(I)                                                BCON0060
      WRITE (KW,902) INTGR(I), REAL(J)                                 BCON0061
    4 CONTINUE                                                         BCCN0062
C LOOP OVER CONSTRAINED DOF FOR MODIFICATICN OF COMPLETELY ASSEMBLED   BCON0063
C FORCE VECTOR                                                         BCON0064
      DO 71 I = ICON,LCON                                              BCON0065
      IF (INTGR(I) .EQ. 0) GO TO 71                                    BCON0066
C CHECK IF PRESCRIBED DISPLACEMENT = 0 -- IF IT DOES, SKIP FORCE VECTOR BCON0067
      MROW = INTGR(I)                                                  BCON0068
      M = IQM1+MROW                                                    BCON0069
      IF (REAL(M) .EQ. 0.) GO TO 71                                    BCON0070
C DISPL .NE. 0 -- LOOP OVER ALL ROWS TC MCDIFY FORCE VECTOR -- SKIP    BCON0071
C CONSTRAINED ROWS (CCNTROLLED BY VALUE OF NEXT)                       BCON0072
```

```
      NEXT = ICON                                                       BCON0073
      DO 7 NROW = 1,NDT                                                 BCON0074
      IF (NROW .NE. INTGR(NEXT)) GO TO 5                                BCON0075
      NEXT = NEXT+1                                                     BCON0076
      GO TO 7                                                           BCON0077
    5 IF (MROW .GT. NROW) GO TO 51                                      BCON0078
C CHECK FOR COUPLING OF ROW NROW WITH COL MROW                           BCON0079
      J = ILNZM1+NROW                                                   BCON0080
      IF (INTGR(J) .GT. MROW) GO TO 7                                   BCON0081
      IROW = NROW                                                       BCON0082
      ICOL = MROW                                                       BCON0083
      GO TO 6                                                           BCON0084
C CHECK FOR COUPLING OF ROW MROW WITH COL NROW                           BCON0085
   51 J = ILNZM1+MROW                                                   BCON0086
      IF (INTGR(J) .GT. NROW) GO TO 7                                   BCON0087
      IROW = MROW                                                       BCON0088
      ICOL = NROW                                                       BCON0089
C SUBTRACT K*(PRESCR DISPL) FROM FORCE VECTOR                            BCON0090
    6 KADR = IKDUM1+IROW                                                BCON0091
      KADR = INTGR(KADR)+ICOL                                           BCON0092
      N = IQM1+NROW                                                     BCON0093
      REAL(N) = REAL(N)-REAL(KADR)*REAL(M)                              BCON0094
    7 CONTINUE                                                          BCON0095
   71 CONTINUE                                                          BCON0096
C LOOP OVER CONSTRAINED ROWS TO DECOUPLE THEM FROM REST OF K MATRIX      BCON0097
      DO 11 I = ICON,LCON                                               BCON0098
      IF (INTGR(I) .EQ. 0) GO TO 11                                     BCON0099
      MROW = INTGR(I)                                                   BCON0100
      INIT = ILNZM1+MROW                                                BCON0101
      INIT = INTGR(INIT)                                                BCON0102
C SET ROW = 0                                                            BCON0103
      M = IKDUM1+MROW                                                   BCON0104
      M = INTGR(M)                                                      BCON0105
      DO 8 MCOL = INIT,MROW                                             BCON0106
      KADR = M+MCOL                                                     BCON0107
    8 REAL(KADR) = 0.                                                   BCON0108
```

```
C SET COLUMN = 0 IN ROWS WHOSE LNZE COL NC IS .LE. MROW -- SKIP THIS     BCON0109
C SECTION IF MROW IS THE LAST ROW                                        BCON0110
      IF (MROW .EQ. NDT) GO TO 10                                        BCON0111
      INIT = MROW+1                                                      BCON0112
      DO 9 NROW = INIT,NDT                                               BCON0113
      N = ILNZM1+NROW                                                    BCON0114
      IF (INTGR(N) .GT. MROW) GO TO 9                                    BCON0115
      KADR = IKOOM1+NROW                                                 BCON0116
      KADR = INTGR(KADR)+MROW                                            BCON0117
      REAL(KADR) = 0.                                                    BCON0118
    9 CONTINUE                                                           BCON0119
C SET DIAGONAL ENTRY = 1                                                 BCON0120
   10 KADR = M+MROW                                                      BCON0121
      REAL(KADR) = 1.                                                    BCON0122
   11 CONTINUE                                                           BCON0123
      RETURN                                                             BCON0124
      END                                                                BCON0125
      SUBROUTINE FACTSD                                                 FACT0001
C***********************************************************************FACT0002
C FINITE ELEMENT ANALYSIS BASIC LIBRARY SUBROUTINE                      FACT0003
C AEROELASTIC AND STRUCTURES RESEARCH LABORATORY                        FACT0004
C MASSACHUSETTS INSTITUTE OF TECHNOLOGY                                 FACT0005
C***********************************************************************FACT0006
C THIS IS ALSO SUBROUTINE FACTPD - SEE ENTRY POINT BELOW                FACT0007
      DIMENSION REAL(10000), INTGR(10000)                               FACT0008
      COMMON /IO/ KR, KW, KP, KT1, KT2, KT3                             FACT0009
      COMMON /SIZE/ NET, NDT                                            FACT0010
      COMMON /BEGIN/ ICON,IKOUNT,ILNZ,IMASTR,IQ,IK                      FACT0011
      COMMON /END/ LCON,LKOUNT,LLNZ,LMASTR,LQ,LK                        FACT0012
      COMMON /DATA/ REAL                                                FACT0013
C VERSION 1 RELEASE 1 AUGUST 1972                                       FACT0014
C                                                                       FACT0015
      EQUIVALENCE (REAL(1), INTGR(1))                                   FACT0016
C PRINT CONTROL                                                         FACT0017
  901 FORMAT (1H0,47X,25HTRIPLE FACTOR ENTRY POINT,/,53H K MATRIX NOT POFACT0018
     2SITIVE-DEFINITE IN THE FOLLOWING ROWS )                           FACT0019
  902 FORMAT (40X,I12)                                                  FACT0020
  903 FORMAT (48X,4HNONE )                                              FACT0021
  904 FORMAT (58H USER SPECIFIED SEMI-DEFINITE MATRIX.  EXECUTION CONTINFACT0022
     2UES )                                                             FACT0023
  905 FORMAT (40H USER SPECIFIED POSITIVE-DEFINITE MATRIX,/,21H EXECUTIOFACT0024
     2N TERMINATES )                                                    FACT0025
  906 FORMAT (25H0K MATRIX SINGULAR IN ROW,I12,/,21H EXECUTION TERMINATEFACT0026
     2S )                                                               FACT0027
  907 FORMAT (61H0LARGEST ROUNDING ERROR IN DIAGONAL FACTORING OCCURRED FACT0028
     2IN ROW,I12,/,43H NUMBER OF LOWEST SIGNIFICANT FIGURES LOST=,I3)   FACT0029
  908 FORMAT (42H ROUNDING ERROR EXCEEDS ACCEPTABLE MAXIMUM,/,21H EXECUTFACT0030
     2ION TERMINATES )                                                  FACT0031
C SET ENTRY FLAG                                                        FACT0032
      NPD = 0                                                           FACT0033
      GO TO 1                                                           FACT0034
      ENTRY FACTPD                                                      FACT0035
      NPD = 1                                                           FACT0036
```

```
C PRINT ENTRY MESSAGE                                                         FACT0037
    1 WRITE (KW,901)                                                          FACT0038
      IKOUM1 = IKOUNT-1                                                       FACT0039
      ILNZM1 = ILNZ-1                                                         FACT0040
C INITIALIZE AT 1ST NONZERO ENTRY IN CONSTRAINT VECTOR                        FACT0041
      DO 2 I = ICON,LCON                                                      FACT0042
      IF (INTGR(I) .EQ. 0) GO TO 2                                            FACT0043
      GO TO 3                                                                 FACT0044
    2 CONTINUE                                                                FACT0045
    3 NEXT = 1                                                                FACT0046
      IF (INTGR(NEXT) .EQ. 1) NEXT = NEXT+1                                   FACT0047
C INITIALIZE ERROR PARAMETERS                                                 FACT0048
      JROW = 1                                                                FACT0049
      MPD = 0                                                                 FACT0050
      TEST = 1.                                                               FACT0051
C DO FIRST ROW AS SPECIAL CASE                                                FACT0052
      IF (REAL(IK) .EQ. 0.) GO TO 14                                          FACT0053
      IF (REAL(IK) .GT. 0.) GO TO 4                                           FACT0054
C NPD MESSAGE AND FLAG                                                        FACT0055
      WRITE (KW,902) JROW                                                     FACT0056
      MPD = 1                                                                 FACT0057
C LOOP OVER REMAINING ROWS                                                    FACT0058
    4 DO 13 MROW = 2,NDT                                                      FACT0059
C CHECK FOR CONSTRAINED ROW - SKIP IF FOUND AND RESET FOR THE NEXT ONE        FACT0060
      IF (MROW .NE. INTGR(NEXT)) GO TO 5                                      FACT0061
      NEXT = NEXT+1                                                           FACT0062
      GO TO 13                                                                FACT0063
C FREE ROW - FACTOR FROM LNZ COL NO TO ROW NO                                 FACT0064
    5 M = ILNZM1+MROW                                                         FACT0065
      M = INTGR(M)                                                            FACT0066
      MM = IKOUM1+MROW                                                        FACT0067
      MM = INTGR(MM)                                                          FACT0068
      DO 12 MCOL = M,MROW                                                     FACT0069
      SUM = 0.                                                                FACT0070
      NN = IKOUM1+MCOL                                                        FACT0071
      NN = INTGR(NN)                                                          FACT0072
```

```
C LNZE IS A SPECIAL CASE - NO SUM REQUIRED                              FACT0073
      IF (MCOL .EQ. M) GO TO 7                                          FACT0074
C START SUM FROM GREATEST OF MROW OR ROW 'MCOL' LNZ COL NOS             FACT0075
      INIT = M                                                          FACT0076
      N = ILNZM1+MCOL                                                   FACT0077
      IF (INTGR(N) .GT. M) INIT = INTGR(N)                              FACT0078
C NO SUM IF ROW 'MCOL' HAS LEADING ZEROS UP TO THE DIAGONAL             FACT0079
      IF (INIT .EQ. MCOL) GO TO 7                                       FACT0080
C ACCUMULATE THE SUM                                                    FACT0081
      LAST = MCOL-1                                                     FACT0082
      DO 6 J = INIT,LAST                                                FACT0083
      KADRM = MM+J                                                      FACT0084
      KADRN = NN+J                                                      FACT0085
      JJ = IKOUM1+J                                                     FACT0086
      JJ = INTGR(JJ)+J                                                  FACT0087
    6 SUM = SUM+REAL(JJ)*REAL(KADRM)*REAL(KADRN)                        FACT0088
C BRANCH TO SPECIAL ALGORITHM FOR DIAGONAL ENTRIES                      FACT0089
    7 IF (MCOL .EQ. MROW) GO TO 8                                       FACT0090
C FOR OFF-DIAGONAL ENTRIES:                                             FACT0091
      KADR = MM+MCOL                                                    FACT0092
      NN = NN+MCOL                                                      FACT0093
      REAL(KADR) = (REAL(KADR)-SUM)/REAL(NN)                            FACT0094
      GO TO 12                                                          FACT0095
C DIAGONAL ENTRY - TEST FOR SINGULARITY AND SEMI-DEFINITENESS           FACT0096
    8 MM = MM+MROW                                                      FACT0097
      IF (REAL(MM)-SUM .NE. 0.) GO TO 9                                 FACT0098
      JROW = MROW                                                       FACT0099
      GO TO 14                                                          FACT0100
    9 IF (REAL(MM)-SUM .GT. 0.) GO TO 10                                FACT0101
      WRITE (KW,902) MROW                                               FACT0102
      MPD = 1                                                           FACT0103
C CALCULATE ROUNDING ERROR                                              FACT0104
   10 TESTR = ABS((REAL(MM)-SUM)/REAL(MM))                              FACT0105
      IF (TESTR .GE. TEST) GO TO 11                                     FACT0106
      TEST = TESTR                                                      FACT0107
      IROW = MROW                                                       FACT0108
```

```
C EVALUATE DIAGONAL ENTRY                                               FACT0109
   11 REAL(MM) = REAL(MM)-SUM                                           FACT0110
   12 CONTINUE                                                          FACT0111
   13 CONTINUE                                                          FACT0112
C SEMI-DEFINITENESS CHECKS AND ROUNDING ERROR OUTPUT                    FACT0113
      IF (MPD .EQ. 0) WRITE (KW,903)                                    FACT0114
      IF (MPD .EQ. 1 .AND. NPD .EQ. 1) WRITE (KW,905)                   FACT0115
      IF (MPD .EQ. 1 .AND. NPD .EQ. 0) WRITE (KW,904)                   FACT0116
      IERR = -1.00001*ALOG10(TEST)                                      FACT0117
      WRITE (KW,907) IROW,IERR                                          FACT0118
      IF (IERR .GT. 5) WRITE (KW,908)                                   FACT0119
      IF (MPD .EQ. 1 .AND. NPD .EQ. 1) STOP                             FACT0120
      IF (IERR .GT. 5) STOP                                             FACT0121
      RETURN                                                            FACT0122
C SINGULAR MATRIX                                                       FACT0123
   14 WRITE (KW,906) JROW                                               FACT0124
      STOP                                                              FACT0125
      END                                                               FACT0126
      SUBROUTINE ORK(LENGTH)
C***********************************************************************
C FINITE ELEMENT ANALYSIS BASIC LIBRARY SUBROUTINE
C AEROELASTIC AND STRUCTURES RESEARCH LABORATORY
C MASSACHUSETTS INSTITUTE OF TECHNOLOGY
C***********************************************************************
C     THIS SUBROUTINE CREATES
C              1) THE LNZ VECTOR WHICH HOLDS THE COLUMN NUMBER
C                 OF THE LEADING NON-ZERO ENTRY IN EACH ROW OF
C                 THE ASSEMBLED "K" MATRIX
C              2) THE ADDRESS COUNT VECTOR WHICH HOLDS THE ABSOLUTE
C                 ADDRESS OF THE DIAGONAL ENTRY FOR EACH ROW OF THE
C                 ASSEMBLED K MATRIX, MINUS THE ROW NUMBER, AND
C                 CHECKS THAT K FITS IN THE /DATA/ VECTOR
      DIMENSION REAL(10000),INTGR(10000)
      COMMON /IO/ KR, KW, KP, KT1, KT2, KT3
      COMMON /SIZE/ NET, NDT
      COMMON /BEGIN/ ICON,IKOUNT,ILNZ,IMASTR,IQ,IK
      COMMON /END/ LCON,LKOUNT,LLNZ,LMASTR,LQ,LK
      COMMON /DATA/ REAL
C VERSION 1 RELEASE 1 AUGUST 1972
C
      EQUIVALENCE (REAL(1),INTGR(1))
C PRINT CONTROL
  100 FORMAT(49H0THE LENGTH OF THE "DATA" VECTOR FOR THIS CASE IS,I10,14
     1H WHICH EXCEEDS,I10,49H =THE MAXIMUM ALLOWED IN THE DIMENSION STAT
     2EMENT./39H EXECUTION TERMINATED IN SUBROUTINE ORK)
  200 FORMAT(5X,3HROW,2X,13HLNZE COL. NO.,2X,18HABS. ADR. OF DIAG.)
  300 FORMAT(I9,I10,I12)
  400 FORMAT(10H0THERE ARE,I10,68H NON-ZERO ENTRIES IN "K". IF "K" WERE
     1FULLY POPULATED THERE WOULD BE,I10, 9H ENTRIES.,/,20X,15HTHE DENSI
     2TY IS ,E15.6)
      ILNZM1=ILNZ-1
      IMSTM1=IMASTR-1
      IKOUM1=IKOUNT-1
      NETM1=NET-1
      IMSTP1=IMASTR+1                                                   ORK 0037
C SET EACH LNZ COLUMN NC = ROW NO (CIAGONAL MATRIX)                     ORK 0038
      DO 30 IROW=1,NDT                                                  ORK 0039
      MSUB=ILNZM1+IROW                                                  ORK 0040
 30   INTGR(MSUB)=IROW                                                  ORK 0041
C EXAMINE MASTER ASSEMBLY LIST, ONE ELEMENT AT A TIME, TO CREATE        ORK 0042
C THE LNZ VECTOR                                                        ORK 0043
      DO 20 LNUM=1,NET                                                  ORK 0044
      MADDR = IMSTM1+LNUM                                               ORK 0045
      MADDR = INTGR(MADDR)-1                                            ORK 0046
C CALCULATE NO. DOF IN THE ELEMENT BY DIFFERENCING POINTERS, OR ...     ORK 0047
      I = IMASTR+LNUM                                                   ORK 0048
      IF(LNUM.EQ.NET) GO TO 3                                           ORK 0049
      NDE = INTGR(I)-INTGR(I-1)                                         ORK 0050
      GO TO 4                                                           ORK 0051
C ... BEGIN BY ASSUMING THE LIST IS FILLED, FOR LAST ELEMENT            ORK 0052
 3    NDE = LMASTR-INTGR(I-1)+1                                         ORK 0053
C INITIALIZE SMALLEST DOF NO. AT LARGEST POSSIBLE VALUE                 ORK 0054
 4    ISMALL=NDT                                                        ORK 0055
C     FIND SMALLEST MASTR NUMBER FOR THIS ELEMENT                       ORK 0056
      DO 5 JDOF=1,NDE                                                   ORK 0057
      INDEX=MADDR+JDOF                                                  ORK 0058
      IF(INTGR(INDEX).GT.ISMALL) GO TO 5                                ORK 0059
C DISCONTINUE SEARCH IF A ZERO IS FCUND, INDICATING EXCESS STORAGE      ORK 0060
C AND PREMATURE END OF LIST FOR LAST ELEMENT                            ORK 0061
      IF(INTGR(INDEX).EQ.0) GO TO 6                                     ORK 0062
      ISMALL=INTGR(INDEX)                                               ORK 0063
 5    CONTINUE                                                          ORK 0064
C     FIND COLUMN NUMBER OF LEADING NCN-ZERO ENTRY IN ROW               ORK 0065
 6    DO 10 JDOF=1,NDE                                                  ORK 0066
      INDEX=MADDR+JDOF                                                  ORK 0067
      INDEX=ILNZM1+INTGR(INDEX)                                         ORK 0068
C CHANGE LNZ COL NO ONLY IF NEW ONE IS LESS THAN OLD ONE                ORK 0069
      IF(INTGR(INDEX).LT.ISMALL) GO TO 8                                ORK 0070
      INTGR(INDEX)=ISMALL                                               ORK 0071
      GO TO 10                                                          ORK 0072
```

```
C DISCONTINUE OPERATION IF EXCESS STORAGE IS DISCOVERED                  ORK 0073
 8    IF(INTGR(INDEX).EQ.0) GO TO 20                                     ORK 0074
 10   CONTINUE                                                           ORK 0075
 20   CONTINUE                                                           ORK 0076
C CREATE ADDRESS COUNT VECTOR                                            ORK 0077
      INTGR(LKOUNT) = IK                                                 ORK 0078
      INDEX=LKOUNT                                                       ORK 0079
      DO 40 IROW=2,NDT                                                   ORK 0080
      I = ILNZM1+IROW                                                    ORK 0081
      INTGR(INDEX+1) = INTGR(INDEX)+IROW+1-INTGR(I)                      ORK 0082
 40   INDEX=INDEX+1                                                      ORK 0083
C ADDRESS COUNT VECTOR NOW CONTAINS ABSOLUTE ADDRESS ONLY FOR THE        ORK 0084
C DIAGONAL ENTRIES, AND THUS INTGR(LKOUNT) = LK EXACTLY                  ORK 0085
      IF (INTGR(LKOUNT) .LE. LENGTH) GO TO 50                            ORK 0086
      WRITE (KW,100) INTGR(LKOUNT), LENGTH                               ORK 0087
      STOP                                                               ORK 0088
 50   LK = INTGR(LKOUNT)                                                 ORK 0089
      WRITE(KW,200)                                                      ORK 0090
      DO 60 IROW=1,NDT                                                   ORK 0091
      I = ILNZM1+IROW                                                    ORK 0092
      J = IKOUM1+IROW                                                    ORK 0093
      WRITE (KW,300) IROW, INTGR(I), INTGR(J)                            ORK 0094
C     REPLACE THE ABS. ADDRESS OF DIAG. BY (ABS. ADDRESS - ROW NO.)      ORK 0095
      INTGR(J) = INTGR(J)-IROW                                           ORK 0096
 60   CONTINUE                                                           ORK 0097
      NENTRY = INTGR(LKOUNT)+NDT-IK+1                                    ORK 0098
      INDEX = (NDT*(NDT+1))/2                                            ORK 0099
      DENS = FLOAT(NENTRY)/FLOAT(INDEX)                                  ORK 0100
      WRITE (KW,400) NENTRY, INDEX, DENS                                 ORK 0101
C ZERO THE FORCE/DISPLACEMENT VECTOR AND THE K MATRIX BLOCK              ORK 0102
      DO 70 I=IQ,LK                                                      ORK 0103
 70   REAL(I)=0.                                                         ORK 0104
      RETURN                                                             ORK 0105
      END                                                                ORK 0106
      SUBROUTINE ROTATE(NODE,IROW,JROW,KRCW,ZANGLE,YANGLE,XANGLE)       ROTA0001
C***********************************************************************ROTA0002
C FINITE ELEMENT ANALYSIS BASIC LIBRARY SUBROUTINE                      ROTA0003
C AEROELASTIC AND STRUCTURES RESEARCH LABCRATORY                        ROTA0004
C MASSACHUSETTS INSTITUTE OF TECHNOLOGY                                 ROTA0005
C***********************************************************************ROTA0006
      DIMENSION REAL(10000),INTGR(10000)                                ROTA0007
      DIMENSION A(3,3), B(3,3), IR(3), INFC(10), KDR(3), PERM(3,3),     ROTA0008
     2 PROTOR(3,3), ROTOR(3,3), TEMPC(3), TEMPR(3), TEMPCC(3), TEMPRR(3)ROTA0009
      COMMON /IO/ KR, KW, KP, KT1, KT2, KT3                             ROTA0010
      COMMON /SIZE/ NET, NDT                                            ROTA0011
      COMMON /BEGIN/ ICON,IKOUNT,ILNZ,IMASTR,IQ,IK                      ROTA0012
      COMMON /DATA/ REAL                                                ROTA0013
C VERSION 1 RELEASE 1 AUGUST 1972                                       ROTA0014
C                                                                       ROTA0015
C                                                                       ROTA0016
      EQUIVALENCE (REAL(1), INTGR(1))                                   ROTA0017
C PRINT CONTROL                                                         ROTA0018
  901 FORMAT(1H0,41X,26HROTATION REQUESTED AT NODE,I10,/,24X,29HASSOCIATROTA0019
     1ED MASTER NUMBERS FOR,8X,5HFIRST,6X,6HSECOND,7X,5HTHIRD,3X,3HDOF,/ROTA0020
     2,54X,3(2X,I10),/,27X,27HEULER ANGLES IN DEGREES ARE,8X,7HTHETA Z,5ROTA0021
     3X,7HTHETA Y,5X,7HTHETA X,/,57X,3(2X,E10.3))                       ROTA0022
  902 FORMAT (1X,79HHALF TRANSFORM (K)*(THETA TRANSPOSE) WAS APPLIED TO ROTA0023
     1THE FOLLOWING DOF BELOW ROW,I10,16H IN THE K MATRIX)              ROTA0024
  903 FORMAT (10(2X,I10))                                               ROTA0025
  904 FORMAT(103H0ABOVE REQUEST INCONSISTENT DUE TO INEQUALITY OF LEADINROTA0026
     1G NON-ZERO ENTRY COLUMN NUMBERS OF ARGUMENT ROWS,/,9X,3HROW,1X,11HROTA0027
     2LNZE COL NO,/,3(2X,I10,2X,I10,/))                                 ROTA0028
  905 FORMAT (1X,41HEXECUTION TERMINATED IN SUBROUTINE ROTATE)          ROTA0029
  906 FORMAT (54H0ABOVE REQUEST INCCNSISTENT DUE TO REPEATED ROW NUMBER)ROTA0030
  907 FORMAT(73H0ABOVE REQUEST INVALID DUE TC INCORRECT LOCATICN OF UNDEROTA0031
     1FINED ROW NUMBERS)                                                ROTA0032
      IR(1)=IROW                                                        ROTA0033
      IR(2)=JROW                                                        ROTA0034
      IR(3)=KROW                                                        ROTA0035
C PRINT ENTRY MESSAGE                                                   ROTA0036
```

```
      NNODE=IABS(NODE)                                                  ROTA0037
      WRITE (KW,901) NNODE, (IR(I), I = 1,3), ZANGLE, YANGLE, XANGLE    ROTA0038
C ESTABLISH DOF ROTATION LIMIT                                          ROTA0039
      LIMIT = 3                                                         ROTA0040
      IF (KROW .LE. 0) LIMIT = 2                                        ROTA0041
      IF(IROW .GT. 0 .AND. JROW .GT. 0) GO TO 100                       ROTA0042
      WRITE(KW,907)                                                     ROTA0043
      WRITE(KW,905)                                                     ROTA0044
      STOP                                                              ROTA0045
C GET LNZE COL NOS AND TEST FOR EQUALITY                                ROTA0046
  100 DO 1 I = 1,LIMIT                                                  ROTA0047
      J = ILNZ+IR(I)-1                                                  ROTA0048
    1 INFO(I) = INTGR(J)                                                ROTA0049
      IF (LIMIT .EQ. 3 .AND. INFO(1) .EQ. INFO(2) .AND. INFO(2) .EQ.    ROTA0050
     2 INFO(3)) GO TO 2                                                 ROTA0051
      IF (LIMIT .EQ. 2 .AND. INFO(1) .EQ. INFO(2)) GO TO 2              ROTA0052
C SOMETHING WRONG - PRINT ERROR MESSAGE                                 ROTA0053
      WRITE (KW,904) (IR(I), INFO(I), I = 1,LIMIT)                      ROTA0054
      WRITE (KW,905)                                                    ROTA0055
      STOP                                                              ROTA0056
C CHECK FOR KEYPUNCH ERROR IN ARGUMENT LIST                             ROTA0057
    2 IF (IROW .NE. JROW .AND. JROW .NE. KROW .AND. KROW .NE. IROW)     ROTA0058
     2 GO TO 3                                                          ROTA0059
      WRITE (KW,906)                                                    ROTA0060
      WRITE (KW,905)                                                    ROTA0061
      STOP                                                              ROTA0062
C OK TO PROCEED - FORM ROTOR MATRIX AFTER CONVERTING                    ROTA0063
C ANGLES TO RADIAN MEASURE                                              ROTA0064
    3 Z = 3.14159*ZANGLE/180.                                           ROTA0065
      Y = 3.14159*YANGLE/180.                                           ROTA0066
      X = 3.14159*XANGLE/180.                                           ROTA0067
      ROTOR(1,1) = COS(Y)*COS(Z)                                        ROTA0068
      ROTOR(1,2) = COS(Y)*SIN(Z)                                        ROTA0069
      ROTOR(1,3) = SIN(Y)                                               ROTA0070
      ROTOR(2,2) = COS(X)*COS(Z)-SIN(X)*SIN(Y)*SIN(Z)                   ROTA0071
      ROTOR(2,3) = SIN(X)*COS(Y)                                        ROTA0072
      ROTOR(3,3) = COS(X)*COS(Y)                                       ROTA0073
      ROTOR(2,1) = -ROTOR(1,2)                                         ROTA0074
      ROTOR(3,1) = -ROTOR(1,3)                                         ROTA0075
      ROTOR(3,2) = -ROTOR(2,3)                                         ROTA0076
C     SKIP K MATRIX SECTION IF A RE-ROTATION                           ROTA0077
      IF(NODE .LT. 0) GO TO 38                                         ROTA0078
C ESTABLISH LEAST AND LARGEST DOF NUMBERS FROM ARGUMENT LIST           ROTA0079
      LEAST = IROW                                                     ROTA0080
      LARGST = JROW                                                    ROTA0081
      DO 4 I = 1,LIMIT                                                 ROTA0082
      IF (IR(I) .LT. LEAST) LEAST = IR(I)                              ROTA0083
      IF (IR(I) .GT. LARGST) LARGST = IR(I)                            ROTA0084
    4 CONTINUE                                                         ROTA0085
C APPLY (ROTOR)*(K) TO ROWS IROW, JROW, KROW - ONE COLUMN AT A TIME    ROTA0086
C FROM INZE TO LEAST-1 -- SKIP SECTION IF LEAST ROW = 1ST ROW          ROTA0087
      INIT = INFO(1)                                                   ROTA0088
      IF (LEAST .EQ. 1 .OR. INIT .EQ. LEAST) GO TO 8                   ROTA0089
      LAST = LEAST-1                                                   ROTA0090
      DO 7 MCOL = INIT,LAST                                            ROTA0091
C GET ENTRIES OF K MATRIX INTO TEMPR, ZERO TEMPC                       ROTA0092
      DO 5 I = 1,LIMIT                                                 ROTA0093
      J = IKOUNT+IR(I)-1                                               ROTA0094
      KADR = INTGR(J)+MCOL                                             ROTA0095
      TEMPR(I) = REAL(KADR)                                            ROTA0096
    5 TEMPC(I) = 0.                                                    ROTA0097
C APPLY TRANSFORM                                                      ROTA0098
      DO 6 I = 1,LIMIT                                                 ROTA0099
      DO 6 J = 1,LIMIT                                                 ROTA0100
    6 TEMPC(I) = TEMPC(I)+ROTOR(I,J)*TEMPR(J)                          ROTA0101
C RESTORE TRANSFORMED COLUMN TO K MATRIX                               ROTA0102
      DO 7 I = 1,LIMIT                                                 ROTA0103
      J = IKOUNT+IR(I)-1                                               ROTA0104
      KADR = INTGR(J)+MCOL                                             ROTA0105
    7 REAL(KADR) = TEMPC(I)                                            ROTA0106
C TEST FOR ROWS INTERVENING BETWEEN IROW, JROW AND KROW                ROTA0107
    8 IF (LARGST .EQ. LEAST+LIMIT-1) GO TO 26                          ROTA0108
```

```
C*********************************************************************ROTA0109
C SPECIAL ALGORITHMS FOR INTERVENING ROWS                               ROTA0110
C*********************************************************************ROTA0111
C PRINT HEADING                                                         ROTA0112
      WRITE (KW,902) LEAST                                              ROTA0113
C FIND VALUE OF MIDDLE ARGUMENT ROW IF 3 DOF ARE BEING ROTATED          ROTA0114
      IF (LIMIT .EQ. 2) GO TO 10                                        ROTA0115
      DO 9 I = 1,3                                                      ROTA0116
      IF (IR(I) .NE. LEAST .AND. IR(I) .NE. LARGST) MIDDLE = IR(I)      ROTA0117
    9 CONTINUE                                                          ROTA0118
C TEST TO SEE IF ARGUMENT ROWS WERE GIVEN IN ASCENDING ORDER.  IF       ROTA0119
C NOT, A PERMUTATION SIMILARITY TRANSFORM MUST BE APPLIED TO THE        ROTA0120
C ROTOR MATRIX TO TRANSFORM THE INTERVENING ROWS PROPERLY               ROTA0121
      IF (IROW .LT. JROW .AND. JROW .LT. KROW) GO TO 15                 ROTA0122
   10 IF (KROW .EQ. 0 .AND. IROW .LT. JROW) GO TO 15                    ROTA0123
C PREPARE STORAGE                                                       ROTA0124
      DO 11 I = 1,LIMIT                                                 ROTA0125
      DO 11 J = 1,LIMIT                                                 ROTA0126
      A(I,J) = 0.                                                       ROTA0127
      PROTOR(I,J) = 0.                                                  ROTA0128
   11 PERM(I,J) = 0.                                                    ROTA0129
      DO 12 I = 1,LIMIT                                                 ROTA0130
      IF (IR(I) .EQ. LEAST) PERM(1,I) = 1.                              ROTA0131
      IF (IR(I) .EQ. LARGST) PERM(LIMIT,I) = 1.                         ROTA0132
      IF (LIMIT .EQ. 2) GO TO 12                                        ROTA0133
      IF (IR(I) .EQ. MIDDLE) PERM(2,I) = 1.                             ROTA0134
   12 CONTINUE                                                          ROTA0135
      DO 13 I = 1,LIMIT                                                 ROTA0136
C APPLY SIMILARITY TRANSFORM                                            ROTA0137
      DO 13 J = 1,LIMIT                                                 ROTA0138
      DO 13 K = 1,LIMIT                                                 ROTA0139
   13 A(I,J) = A(I,J)+PERM(I,K)*ROTOR(K,J)                              ROTA0140
      DO 14 I = 1,LIMIT                                                 ROTA0141
      DO 14 J = 1,LIMIT                                                 ROTA0142
      DO 14 K = 1,LIMIT                                                 ROTA0143
   14 PROTOR(I,J) = PROTOR(I,J)+A(I,K)*PERM(J,K)                        ROTA0144
```

```
      GO TO 17                                                          ROTA0145
C PERMUTATION NOT REQUIRED                                              ROTA0146
   15 DO 16 I = 1,3                                                     ROTA0147
      DO 16 J = 1,3                                                     ROTA0148
   16 PROTOR(I,J) = KOTOR(I,J)                                          ROTA0149
C INITIALIZE INFO VECTOR FILL INDEX AND ESTABLISH LOOP LIMITS FOR       ROTA0150
C INTERVENING ROWS                                                      ROTA0151
   17 INFOX = 0                                                         ROTA0152
      INIT = LEAST+1                                                    ROTA0153
C LAST ROW TO BE DONE DEPENDS ON WHETHER 2 OR 3 DOF ARE BEING ROTATED   ROTA0154
      IF (LIMIT .EQ. 2) LAST = LARGST-1                                 ROTA0155
      IF (LIMIT .EQ. 3) LAST = MIDDLE-1                                 ROTA0156
C LOOP OVER REMAINING COLUMNS, (PROTOR)*(K), AND FIRST ROWS,            ROTA0157
C (K)*(PROTOR-T), FROM INIT TO LAST -- SKIP IF MIDDLE DOF = LEAST+1     ROTA0158
C WHEN 3 DOF ARE BEING ROTATED                                          ROTA0159
      IF (LIMIT .EQ. 3 .AND. LAST .EQ. LEAST) GO TO 25                  ROTA0160
C THIS IS THE RE-ENTRY POINT IF 3 DOF ARE BEING ROTATED AND ROWS        ROTA0161
C INTERVENE BETWEEN MIDDLE AND LARGEST                                  ROTA0162
   18 DO 24 MROW = INIT,LAST                                            ROTA0163
C CHECK LNZE COL NO OF CURRENT ROW -- IF .GT. LOWER ROTATED DOF,        ROTA0164
C WHICH = INIT-1, K MATRIX CONTAINS STORED OR NON-STORED ZEROS AND      ROTA0165
C BOTH ROW AND COLUMN TRANSFORM CAN BE SKIPPED                          ROTA0166
      J = ILNZ+MROW-1                                                   ROTA0167
      IF (INTGR(J) .GT. INIT-1) GO TO 24                                ROTA0168
C INCREMENT INFO FILL INDEX AND STORE CURRENT ROW NUMBER                ROTA0169
      INFOX = INFOX+1                                                   ROTA0170
      INFO(INFOX) = MROW                                                ROTA0171
C IF TEN ROW NUMBERS HAVE BEEN COLLECTED, PRINT THEM AND RESET INDEX    ROTA0172
      IF (INFOX .LT. 10) GO TO 19                                       ROTA0173
      WRITE (KW,903) (INFO(J), J = 1,10)                                ROTA0174
      INFOX = 0                                                         ROTA0175
C GET CORRECT ABSOLUTE ADDRESSES FOR K MATRIX ENTRIES, REPLACING COLUMN ROTA0176
C SECTORS ABOVE LOWER TRIANGLE BY ROW SECTORS BELOW, AND VICE-VERSA     ROTA0177
   19 J = IKOUNT+MROW-1                                                 ROTA0178
      KDR(1) = INTGR(J)                                                 ROTA0179
      J = IKOUNT+LAST                                                   ROTA0180
      KDR(2) = INTGR(J)+MROW                                            ROTA0181
      IF (INIT .EQ. LEAST+1) GO TO 20                                   ROTA0182
      KDR(2) = KDR(1)+INIT-1                                            ROTA0183
   20 KDR(1) = KDR(1)+LEAST                                             ROTA0184
      IF (LIMIT .EQ. 2) GO TO 21                                        ROTA0185
      J = IKOUNT+LARGST-1                                               ROTA0186
      KDR(3) = INTGR(J)+MROW                                            ROTA0187
C PICK UP K MATRIX ENTRIES IN TEMPC, TEMPR AND ZERO TEMPCC, TEMPRR      ROTA0188
   21 DO 22 I = 1,LIMIT                                                 ROTA0189
      KADR = KDR(I)                                                     ROTA0190
      TEMPC(I) = REAL(KADR)                                             ROTA0191
      TEMPR(I) = REAL(KADR)                                             ROTA0192
      TEMPCC(I) = 0.                                                    ROTA0193
   22 TEMPRR(I) = 0.                                                    ROTA0194
C APPLY TRANSFORMS (PROTOR)*(TEMPC) AND (TEMPR)*(PROTOR-TRANSPOSE)      ROTA0195
      DO 23 I = 1,LIMIT                                                 ROTA0196
      DO 23 J = 1,LIMIT                                                 ROTA0197
      TEMPCC(I) = TEMPCC(I)+PROTOR(I,J)*TEMPC(J)                        ROTA0198
   23 TEMPRR(I) = TEMPRR(I)+TEMPR(J)*PROTOR(I,J)                        ROTA0199
C RESTORE TRANSFORMED VALUES TO K MATRIX                                ROTA0200
      KADR = KDR(1)                                                     ROTA0201
      REAL(KADR) = TEMPCC(1)                                            ROTA0202
      KADR = KDR(2)                                                     ROTA0203
      IF (INIT .EQ. LEAST+1) REAL(KADR) = TEMPRR(2)                     ROTA0204
      IF (INIT .NE. LEAST+1) REAL(KADR) = TEMPCC(2)                     ROTA0205
      IF (LIMIT .EQ. 2) GO TO 24                                        ROTA0206
      KADR = KDR(3)                                                     ROTA0207
      REAL(KADR) = TEMPRR(3)                                            ROTA0208
   24 CONTINUE                                                          ROTA0209
C PRINT PARTIALLY FILLED INFO VECTOR IF THERE IS AT LEAST ONE ROW,      ROTA0210
C RESET INDEX                                                           ROTA0211
      IF (INFOX .EQ. 0) GO TO 25                                        ROTA0212
      WRITE (KW,903) (INFO(I), I = 1,INFOX)                             ROTA0213
      INFOX = 0                                                         ROTA0214
C COMPLETION TEST                                                       ROTA0215
   25 IF (LIMIT .EQ. 2 .OR. INIT .NE. LEAST+1) GO TO 26                 RCTA0216
```

```
C CHECK FOR INTERVENING ROWS BETWEEN MIDDLE AND LARGEST                        ROTA0217
      IF (MIDDLE+1 .EQ. LARGST) GO TO 26                                         ROTA0218
C RESET LOOP LIMITS, PRINT NEW HEADING                                          ROTA0219
      INIT = MIDDLE+1                                                            ROTA0220
      LAST = LARGST-1                                                            ROTA0221
      WRITE (KW,902) MIDDLE                                                      ROTA0222
      GO TO 16                                                                   ROTA0223
C***********************************************************************ROTA0224
C END OF SPECIAL ALGORITHM SECTION                                               ROTA0225
C***********************************************************************ROTA0226
C FOR 2X2 OR 3X3 K MATRIX ENTRIES DIRECTLY ASSOCIATED WITH DOF BEING             ROTA0227
C ROTATED, APPLY FULL SIMILARITY TRANSFORM                                       ROTA0228
   26 DO 28 I = 1,LIMIT                                                          ROTA0229
      DO 28 J = 1,I                                                              ROTA0230
      K = IKOUNT+IR(I)-1                                                         ROTA0231
      KADR = INTGR(K)+IR(J)                                                      ROTA0232
      IF (IR(I) .GT. IR(J)) GO TO 27                                             ROTA0233
      K = IKOUNT+IR(J)-1                                                         ROTA0234
      KADR = INTGR(K)+IR(I)                                                      ROTA0235
C STORE K MATRIX ENTRY IN A, ZERO B AND SYMMETRIZE                               ROTA0236
   27 A(I,J) = REAL(KADR)                                                        ROTA0237
      B(I,J) = 0.                                                                ROTA0238
      A(J,I) = A(I,J)                                                            ROTA0239
   28 B(J,I) = 0.                                                                ROTA0240
C APPLY HALF OF TRANSFORM, (ROTOR)*(A)                                           ROTA0241
      DO 29 I = 1,LIMIT                                                          ROTA0242
      DO 29 J = 1,LIMIT                                                          ROTA0243
      DO 29 K = 1,LIMIT                                                          ROTA0244
   29 B(I,J) = B(I,J)+ROTOR(I,K)*A(K,J)                                          ROTA0245
C PREPARE STORAGE                                                                ROTA0246
      DO 30 I = 1,LIMIT                                                          ROTA0247
      DO 30 J = 1,LIMIT                                                          ROTA0248
   30 A(I,J) = 0.                                                                ROTA0249
C APPLY 2ND HALF OF TRANSFORM, (B)*(ROTOR-TRANSPOSE)                             ROTA0250
      DO 31 I = 1,LIMIT                                                          ROTA0251
      DO 31 J = 1,LIMIT                                                          ROTA0252
      DO 31 K = 1,LIMIT                                                 ROTA0253
   31 A(I,J) = A(I,J)+B(I,K)*ROTOR(J,K)                                 ROTA0254
C RESTORE TRANSFORMED VALUES TO K MATRIX                                ROTA0255
      DO 32 I = 1,LIMIT                                                 ROTA0256
      DO 32 J = 1,I                                                     ROTA0257
      K = IKOUNT+IR(I)-1                                                ROTA0258
      KADR = INTGR(K)+IR(J)                                             ROTA0259
      IF (IR(I) .GE. IR(J)) GO TO 32                                    ROTA0260
      K = IKOUNT+IR(J)-1                                                ROTA0261
      KADR = INTGR(K)+IR(I)                                             ROTA0262
   32 REAL(KADR) = A(I,J)                                               ROTA0263
C APPLY (K)*(ROTOR-TRANSPOSE) TO ROWS BELCW LARGEST IF LNZE COL NOS     ROTA0264
C ARE .LE. LEAST. (IF LNZE COL NO IS .GT. LEAST, THE ROW CONTAINS       ROTA0265
C STORED OR NON-STORED ZEROS IN CCLUMNS IROW, JROW, KROW AND CAN        ROTA0266
C BE SKIPPED.) -- SKIP THIS SECTICN IF LARGEST DOF = NDT                ROTA0267
      IF (LARGST .EQ. NDT) GO TO 38                                     ROTA0268
C PRINT NEW HEADING, RESET LOOP LOWER LIMIT AND INFO INDEX              ROTA0269
      WRITE (KW,902) LARGST                                             ROTA0270
      INIT = LARGST+1                                                   ROTA0271
      INFOX = 0                                                         ROTA0272
      DO 37 MROW = INIT,NDT                                             ROTA0273
C CHECK LNZE COL NO                                                     ROTA0274
      J = ILNZ+MROW-1                                                   ROTA0275
      IF (INTGR(J) .GT. LEAST) GO TO 37                                 ROTA0276
C INCREMENT INFO INDEX, STORE ROW NUMBER                                ROTA0277
      INFOX = INFOX+1                                                   ROTA0278
      INFO(INFOX) = MROW                                                ROTA0279
C IF INFO VECTOR IS FILLED, PRINT AND RESET INDEX                       ROTA0280
      IF (INFOX .LT. 10) GO TO 33                                       ROTA0281
      WRITE (KW,903) (INFO(I), I = 1,10)                                ROTA0282
      INFOX = 0                                                         ROTA0283
C GET ENTRIES OF K MATRIX INTO TEMPC, ZERC TEMPR                        ROTA0284
   33 K = IKOUNT+MROW-1                                                 ROTA0285
      K = INTGR(K)                                                      ROTA0286
      DO 34 I = 1,LIMIT                                                 ROTA0287
      KADR = K+IR(I)                                                    ROTA0288
      TEMPC(I) = REAL(KADR)                                           ROTA0289
   34 TEMPR(I) = 0.                                                   ROTA0290
C APPLY TRANSFORM                                                     ROTA0291
      DO 35 I = 1,LIMIT                                               ROTA0292
      DO 35 J = 1,LIMIT                                               ROTA0293
   35 TEMPR(I) = TEMPR(I)+TEMPC(J)*ROTOR(I,J)                         ROTA0294
C RESTORE TO K                                                        ROTA0295
      DO 36 I = 1,LIMIT                                               ROTA0296
      KADR = K+IR(I)                                                  ROTA0297
   36 REAL(KADR) = TEMPR(I)                                           ROTA0298
   37 CONTINUE                                                        ROTA0299
C CHECK FOR PARTIALLY FILLED INFO VECTOR                              ROTA0300
      IF (INFOX .EQ. 0) GO TO 38                                      ROTA0301
      WRITE (KW,903) (INFO(I), I = 1,INFOX)                           ROTA0302
C APPLY TRANSFORM (ROTOR)*(Q) TO ASSEMBLED ELEMENT EQUIVALENT         ROTA0303
C NODAL FORCES                                                        ROTA0304
   38 DO 39 I = 1,LIMIT                                               ROTA0305
      J = IQ+IR(I)-1                                                  ROTA0306
      TEMPR(I) = REAL(J)                                              ROTA0307
   39 TEMPC(I) = 0.                                                   ROTA0308
      DO 40 I = 1,LIMIT                                               ROTA0309
      DO 40 J = 1,LIMIT                                               ROTA0310
   40 TEMPC(I) = TEMPC(I)+ROTOR(I,J)*TEMPR(J)                         ROTA0311
      DO 41 I = 1,LIMIT                                               ROTA0312
      J = IQ+IR(I)-1                                                  ROTA0313
   41 REAL(J) = TEMPC(I)                                              ROTA0314
      RETURN                                                          ROTA0315
      END                                                             ROTA0316
      SUBROUTINE SETUP(LENGTH,NCON,MASTRL)                              SETU0001
C***********************************************************************SETU0002
C FINITE ELEMENT ANALYSIS BASIC LIBRARY SUBROUTINE                      SETU0003
C AERUELASTIC AND STRUCTURES RESEARCH LABORATORY                        SETU0004
C MASSACHUSETTS INSTITUTE OF TECHNOLOGY                                 SETU0005
C***********************************************************************SETU0006
      DIMENSION REAL(10000),INTGR(10000)                                SETU0007
      DIMENSION II(6), LI(6), KD(6)                                     SETU0008
      COMMON /IO/ KR, KW, KP, KT1, KT2, KT3                             SETU0009
      COMMON /SIZE/ NET, NDT                                            SETU0010
      COMMON /BEGIN/ ICON,IXOUNT,ILNZ,IMASTR,IQ,IK                      SETU0011
      COMMON /END/ LCON,LKOUNT,LLNZ,LMASTR,LQ,LK                        SETU0012
      COMMON /DATA/ REAL                                                SETU0013
C VERSION 1 RELEASE 1 AUGUST 1972                                       SETU0014
      EQUIVALENCE (REAL(1), INTGR(1))                                   SETU0015
      EQUIVALENCE (ICON,II(1)), (LCON,LI(1)), (KR,KD(1))                SETU0016
C PRINT CONTROL                                                         SETU0017
  901 FORMAT (1H0,53X,11HSETUP ENTRY,/,42X,35HUSER SPECIFICATIONS TO FEASETU0018
     1BL SYSTEM,/,1H0,51X,16HI/O DEVICE CODES,/,27X,6HREADER,5X,7HPRINTESETU0019
     2R,2X,10HCARD PUNCH,7X,5HTAPE1,7X,5HTAPE2,7X,5HTAPE3,/,21X,6(2X,I10SETU0020
     3),/,1H0,53X,12HPROBLEM SIZE,/,42X,25HTOTAL NUMBER OF ELEMENTS=,I10SETU0021
     4,/,42X,25HTOTAL DEGREES OF FREEDOM=,I10,/,1H0,51X,16HENTRY PARAMETSETU0022
     5ERS,/,39X,32HASSUMED LENGTH OF /DATA/ VECTOR=,I10,/,32X,45HNUMBER SETU0023
     6OF DISPLACEMENT CONSTRAINTS REQUESTED=,I10,/,33X,44HNUMBER OF WORDSETU0024
     7S REQUESTED FOR ASSEMBLY LIST=,I10)                               SETU0025
  902 FORMAT (1H0,43X,31H/DATA/ VECTOR ADDRESS INDEX MAP,/,22X,11HCONSTRSETU0026
     1AINTS,2X,10HDG ABS ADR,1X,11HLNZE COL NO,1X,11HASMBLY LIST,2X,10HQSETU0027
     2/U VECTOR,4X,8HK MATRIX,/,16X,5HBEGIN,6(2X,I10),/,18X,3HEND,5(2X,ISETU0028
     310),11X,1H*,/,1H0,36H* LK IS CALCULATED BY SUBROUTINE ORK)        SETU0029
  903 FORMAT (40HOSTORAGE EXCEEDS LENGTH OF /DATA/ VECTOR,/,1X,34HLENGTHSETU0030
     1 SUGGESTED FOR THIS PROBLEM=,I12,6H WORDS,/,1X,40HEXECUTION TERMINSETU0031
     2ATED IN SUBROUTINE SETUP)                                         SETU0032
C **********************************************************************SETU0033
C REMOVE THIS FORMAT AND WRITE STATEMENT INDICATED BELOW IF FEABL       SETU0034
C HEADING IS NOT DESIRED                                                SETU0035
 1001 FORMAT(1H1/3(1H ,92(1HX)/),4(5H XXXX,84X,4HXXXX/),5H XXXX,7X,10(1HSETU0036
     1F),5X,10(1HE),9X,2HAA,9X,8(1HB),7X,3HLLL,14X,4HXXXX/5H XXXX,7X,10(SETU0037
     21HF),5X,10(1HE),8X,4HAAAA,8X,9(1HB),6X,3HLLL,14X,4HXXXX/5H XXXX,7XSETU0038
     3,3HFFF,12X,3HEEE,14X,6(1HA),7X,3HBBB,3X,4HBBBB,5X,3HLLL,14X,4HXXXXSETU0039
     4/5H XXXX,7X,3HFFF,12X,3HEEE,13X,3HAAA,2X,3HAAA,6X,3HBBB,4X,3HBBB,5SETU0040
     5X,3HLLL,14X,4HXXXX/5H XXXX,7X,3HFFF,12X,3HEEE,12X,3HAAA,4X,3HAAA,5SETU0041
     6X,3HBBB,4X,3HBBB,5X,3HLLL,14X,4HXXXX/5H XXXX,7X,3HFFF,12X,3HEEE,12SETU0042
     7X,3HAAA,4X,3HAAA,5X,3HBBB,3X,4HBBBB,5X,3HLLL,14X,4HXXXX/5H XXXX,7XSETU0043
     8,8(1HF),7X,8(1HE),7X,3HAAA,4X,3HAAA,5X,9(1HB),6X,3HLLL,14X,4HXXXX/SETU0044
     95H XXXX,7X,8(1HF),7X,8(1HE),7X,10(1HA),5X,9(1HB),6X,3HLLL,14X,4HXXSETU0045
     AXX/5H XXXX,7X,3HFFF,12X,3HEEE,12X,10(1HA),5X,3HBBB,3X,4HBBBB,5X,3HSETU0046
     BLLL,14X,4HXXXX/5H XXXX,7X,3HFFF,12X,3HEEE,12X,3HAAA,4X,3HAAA,5X,3HSETU0047
     CBBB,4X,3HBBB,5X,3HLLL,14X,4HXXXX/5H XXXX,7X,3HFFF,12X,3HEEE,12X,3HSETU0048
     DAAA,4X,3HAAA,5X,3HBBB,3X,4HBBBB,5X,3HLLL,14X,4HXXXX/5H XXXX,7X,3HFSETU0049
     EFF,12X,10(1HE),5X,3HAAA,4X,3HAAA,5X,9(1HB),6X,10(1HL),7X,4HXXXX/5HSETU0050
     F XXXX,7X,3HFFF,12X,10(1HE),5X,3HAAA,4X,3HAAA,5X,8(1HB),7X,10(1HL),SETU0051
     G7X,4HXXXX/4(5H XXXX,84X,4HXXXX/),5H XXXX,24X,37HFINITE ELEMENT ANASETU0052
     HLYSIS BASIC LIBRARY,23X,4HXXXX/4(5H XXXX,84X,4HXXXX/),3(1H ,92(1HXSETU0053
     I)/),1H1)                                                          SETU0054
C **********************************************************************SETU0055
C PRINT ENTRY MESSAGE                                                   SETU0056
C **********************************************************************SETU0057
C REMOVE THIS WRITE STATEMENT IF FEABL HEADING IS NOT DESIRED           SETU0058
      WRITE (KW,1001)                                                   SETU0059
C **********************************************************************SETU0060
      WRITE (KW,901) (KD(I), I = 1,6), NET, NDT, LENGTH, NCON, MASTRL   SETU0061
C CALCULATE ADDRESS INDEX VALUES                                        SETU0062
      ICON = 1                                                          SETU0063
      LCON = NCON                                                       SETU0064
      IKOUNT = LCON+1                                                   SETU0065
      LKOUNT = LCON+NDT                                                 SETU0066
      ILNZ = LKOUNT+1                                                   SETU0067
      LLNZ = LKOUNT+NDT                                                 SETU0068
      IMASTR = LLNZ+1                                                   SETU0069
      LMASTR = LLNZ+MASTRL                                              SETU0070
      IQ = LMASTR+1                                                     SETU0071
      LQ = LMASTR+NDT                                                   SETU0072
      IK = LQ+1                                                         SETU0073
C PRINT INDEX MAP                                                       SETU0074
      WRITE (KW,902) (II(I), I = 1,6), (LI(I), I = 1,5)                 SETU0075
C STORAGE BOUNDS TEST                                                   SETU0076
      IF (LMASTR .LE. LENGTH) GO TO 1                                   SETU0077
C STORAGE EXCEEDED - ESTIMATE REQUIRED LENGTH BASED ON LOWER TRIANGLE   SETU0078
C ESTIMATED POPULATION FACTOR                                           SETU0079
      TR = (NDT*(NDT+1))/2                                              SETU0080
      DENS = 0.5                                                        SETU0081
      IF (NDT .GT. 200) DENS=0.3                                        SETU0082
      IF (NDT .GT. 500) DENS=0.2                                        SETU0083
      IF (NDT .GT. 1500) DENS = 0.15                                    SETU0084
      IF (NDT .GT. 2000) DENS = 0.10                                    SETU0085
      TR = TR*DENS                                                      SETU0086
      LENGTH = LQ+TR                                                    SETU0087
      WRITE (KW,903) LENGTH                                             SETU0088
      STOP                                                              SETU0089
C ENOUGH STORAGE EXISTS TO GO THRU CRK                                  SETU0090
    1 DO 2 I = ICON,LCON                                                SETU0091
    2 INTGR(I) = 0                                                      SETU0092
      RETURN                                                            SETU0093
      END                                                               SETU0094
      SUBROUTINE SIMULQ(ENERGY)
C***********************************************************************
C FINITE ELEMENT ANALYSIS BASIC LIBRARY SUBRCUTINE
C AEROELASTIC AND STRUCTURES RESEARCH LABCRATORY
C MASSACHUSETTS INSTITUTE OF TECHNCLCGY
C***********************************************************************
      DIMENSION REAL(10000), INTGR(10000)
      COMMON /IO/ KR, KW, KP, KT1, KT2, KT3
      COMMON /SIZE/ NET, NDT
      COMMON /BEGIN/ ICCN, IKOUNT, ILNZ, IMASTR, IQ, IK
      COMMON /END/ LCON,LKOUNT,LLNZ,LMASTR,LQ,LK
      COMMON /DATA/ REAL
C VERSION 1 RELEASE 1 AUGUST 1972
C
      EQUIVALENCE (REAL(1), INTGR(1))
C PRINT CONTROL
  901 FORMAT (1H0,53X,18HSIMULQ ENTRY POINT,/,38H PRESCRIBED FORCE/DISPL
     2ACEMENT VECTOR: )
  902 FORMAT (30H0DISPLACEMENT SOLUTICN VECTOR: )
  903 FORMAT (1H0,2X,3HROW,10I12)
  904 FORMAT (6H VALUE,10(2X,E10.3))
  905 FORMAT (32H0STRAIN ENERGY IN THE STRUCTURE=,E10.3)
      IKOUM1 = IKOUNT-1
      ILNZM1 = ILNZ-1
      IQM1 = IQ-1
C PRINT ENTRY MESSAGE, SET PRINT SECTION CONTROL FLAG
      WRITE (KW,901)
      IFLAG = 0
C PRINT SECTION:  DIVIDE Q VECTOR INTC GRCUPS OF TEN
C PLUS A POSSIBLE REMAINDER
      MOST = NDT/10
      LEFT = NDT-10*MOST
      JBEG = 1
      NBEG = IQ
C RE-ENTRY POINT FOR SOLUTION OUTPUT
    1 IF (NDT .LT. 10) GO TO 3
```

```
      DO 2 I = 1,MOST                                                   SIMU0037
      JBEG = 1+10*(I-1)                                                 SIMU0038
      JEND = JBEG+9                                                     SIMU0039
      NBEG = IQ+JBEG-1                                                  SIMU0040
      NEND = NBEG+9                                                     SIMU0041
      WRITE (KW,903) (J, J = JBEG,JEND)                                 SIMU0042
    2 WRITE (KW,904) (REAL(N), N = NBEG,NEND)                           SIMU0043
C CHECK FOR EXISTENCE OF REMAINDER                                      SIMU0044
      IF (LEFT .EQ. 0) GO TO 4                                          SIMU0045
      JBEG = JEND+1                                                     SIMU0046
      NBEG = NEND+1                                                     SIMU0047
    3 WRITE (KW,903) (J, J = JBEG,NDT)                                  SIMU0048
      WRITE (KW,904) (REAL(N), N = NBEG,LC)                             SIMU0049
C CHECK CONTROL FLAG                                                    SIMU0050
    4 IF (IFLAG .EQ. 0) GO TO 5                                         SIMU0051
      WRITE (KW,905) ENERGY                                             SIMU0052
      RETURN                                                            SIMU0053
C FORWARD SOLUTION - NO DIVISIONS, SO SKIP FIRST ROW ENTIRELY           SIMU0054
C INITIALIZE CONSTRAINT VECTOR AT 1ST NONZERO ENTRY                     SIMU0055
    5 DO 6 I = ICON,LCCN                                                SIMU0056
      IF (INTGR(I) .EQ. 0) GO TO 6                                      SIMU0057
      GO TO 7                                                           SIMU0058
    6 CONTINUE                                                          SIMU0059
    7 NEXT = I                                                          SIMU0060
      IF (INTGR(NEXT) .EQ. 1) NEXT = NEXT+1                             SIMU0061
C SOLVE (A)(R) = (Q)                                                    SIMU0062
      DO 10 MROW = 2,NDT                                                SIMU0063
C CHECK FOR CONSTRAINT TO SKIP ROW                                      SIMU0064
      IF (MROW .NE. INTGR(NEXT)) GO TO 8                                SIMU0065
C UPDATE NEXT                                                           SIMU0066
      NEXT = NEXT+1                                                     SIMU0067
      GO TO 10                                                          SIMU0068
C INITIALIZE SUM AND LOOP LIMITS FOR SUM                                SIMU0069
    8 SUM = 0.                                                          SIMU0070
      LAST = MROW-1                                                     SIMU0071
      M = ILNZ+LAST                                                     SIMU0072
      M = INTGR(4)                                                      SIMU0073
      MM = IKOUNT+LAST                                                  SIMU0074
      MM = INTGR(MM)                                                    SIMU0075
      DO 9 J = M,LAST                                                   SIMU0076
      KADR = MM+J                                                       SIMU0077
      N = IQM1+J                                                        SIMU0078
    9 SUM = SUM+REAL(KADR)*REAL(N)                                      SIMU0079
C SUBTRACT SUM FROM Q                                                   SIMU0080
      N = IQM1+MROW                                                     SIMU0081
      REAL(N) = REAL(N)-SUM                                             SIMU0082
   10 CONTINUE                                                          SIMU0083
C SOLVE (D)(P) = (R) AND CALCULATE ENERGY                               SIMU0084
      N = IQ-1                                                          SIMU0085
      ENERGY = 0.                                                       SIMU0086
      DO 11 MROW = 1,NDT                                                SIMU0087
      KADR = IKOUM1+MROW                                                SIMU0088
      KADR = INTGR(KADR)+MROW                                           SIMU0089
      N = N+1                                                           SIMU0090
      REAL(N) = REAL(N)/REAL(KADR)                                      SIMU0091
   11 ENERGY = ENERGY+REAL(KADR)*(REAL(N)**2)                           SIMU0092
      ENERGY = 0.5*ENERGY                                               SIMU0093
C BACK SOLUTION - NO DIVISIONS, SO SKIP LAST ROW ENTIRELY               SIMU0094
      NEXT = LCON                                                       SIMU0095
      IF (INTGR(NEXT) .EQ. NDT) NEXT = NEXT-1                           SIMU0096
C LOOP BACKWARDS OVER REMAINING ROWS                                    SIMU0097
      DO 14 I = 2,NDT                                                   SIMU0098
      MROW = NDT+1-I                                                    SIMU0099
C CHECK FOR CONSTRAINT TO SKIP ROW                                      SIMU0100
      IF (MROW .NE. INTGR(NEXT)) GO TO 12                               SIMU0101
C UPDATE NEXT                                                           SIMU0102
      NEXT = NEXT-1                                                     SIMU0103
      IF (NEXT .LT. ICON) NEXT = LCON                                   SIMU0104
      GO TO 14                                                          SIMU0105
C INITIALIZE SUM AND LOWER LOOP LIMIT                                   SIMU0106
   12 SUM = 0.                                                          SIMU0107
      INIT = MROW+1                                                     SIMU0108
      DO 13 J = INIT,NDT                                                SIMU0109
C CHECK IF LNZ COL NO OF ROW J EXCEEDS MRCW - IF SO, SKIP                SIMU0110
      N = ILNZM1+J                                                      SIMU0111
      IF (INTGR(N) .GT. MROW) GO TO 13                                  SIMU0112
      KADR = IKOJM1+J                                                   SIMU0113
      KADR = INTGR(KADR)+MROW                                           SIMU0114
      N = IQM1+J                                                        SIMU0115
      SUM = SUM+REAL(KADR)*REAL(N)                                      SIMU0116
   13 CONTINUE                                                          SIMU0117
C SUBTRACT SUM FROM Q                                                   SIMU0118
      N = IQM1+MROW                                                     SIMU0119
      REAL(N) = REAL(N)-SUM                                             SIMU0120
   14 CONTINUE                                                          SIMU0121
C PRINT SOLUTION HEADING, RESET CONTROL FLAG AND                        SIMU0122
C BRANCH TO OUTPUT SECTION                                              SIMU0123
      WRITE (KW,902)                                                    SIMU0124
      IFLAG = 1                                                         SIMU0125
      GO TO 1                                                           SIMU0126
      END                                                               SIMU0127
```